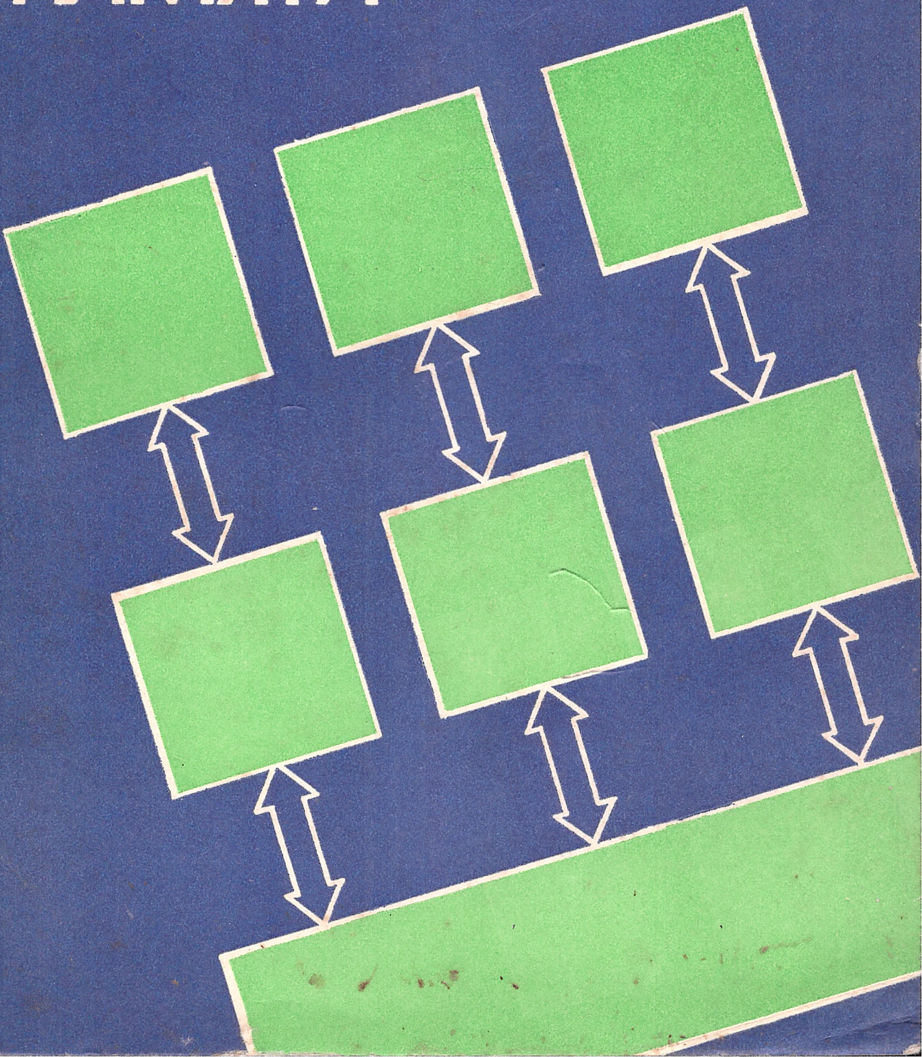
А. ПОМ, О. АГРАВАЛ

БЫСТРО-ДЕЙСТВУЮЩИЕ СИСТЕМЫ ПАМЯТИ



БЫСТРОДЕЙСТВУЮЩИЕ СИСТЕМЫ ПАМЯТИ

ИЗДАТЕЛЬСТВО МИР

БЫСТРО-
ДЕЙСТВУЮЩИЕ
СИСТЕМЫ ПАМЯТИ

A. V. POHM, O. P. AGRAWAL

HIGH-SPEED MEMORY SYSTEMS

RESTON PUBLISHING COMPANY, INC., RESTON, VIRGINIA
A PRENTICE — HALL COMPANY
1983

А. ПОМ, О. АГРАВАЛ

БЫСТРО-ДЕЙСТВУЮЩИЕ СИСТЕМЫ ПАМЯТИ

Перевод с английского
канд. техн. наук В. М. КИСЕЛЬНИКОВА
канд. техн. наук А. В. ЛАПШИНА

под редакцией
д-ра техн. наук В. И. Зуева

Москва «Мир» 1987

ББК 32.973
П 55
УДК 681.3

Пом А., Агравал О.

П 55 Быстродействующие системы памяти: Пер. с англ. — М.: Мир, 1987. — 264 сл., ил.

Книга известных американских специалистов посвящена методике проектирования и определения основных характеристик сверхбыстродействующих устройств памяти современных ЭВМ. Рассматриваются вопросы организации работы многоуровневой памяти. Особое внимание уделяется построению кэш-памяти. Приводятся конкретные структурные схемы памяти и их параметры для некоторых типов вычислительных машин.

Для научных работников и инженеров, занимающихся вопросами разработки запоминающих устройств ЭВМ, а также для аспирантов и студентов, изучающих вычислительную технику.

П $\frac{2405000000-060}{041(01)-87}$ 175—86, ч. 1 **ББК 32.973**

Редакция литературы по информатике и робототехнике

Монография

А. Пом, О. Агравал

БЫСТРОДЕЙСТВУЮЩИЕ СИСТЕМЫ ПАМЯТИ

Старший научный редактор Л. П. Якименко. Младший научный редактор Н. И. Сивилева. Художник Г. М. Чеховский. Художественный редактор Н. М. Иванов. Технический редактор А. Г. Резоухова. Корректор В. И. Киселева.

ИБ № 5589

Сдано в набор 15.02.86. Подписано к печати 21.10.86. Формат 60×90¹/₁₆. Бумага кн.-журнальная. имп. Печать высокая. Гарнитура литературная. Объем 8.25 бум. л. Усл. печ. л. 16.50. Усл. кр.-отт. 16.76. Уч.-изд. л. 16.01. Изд. № 6/4221. Тираж 10 000 экз. Зак. 99. Цена 1 р. 20 к.

ИЗДАТЕЛЬСТВО «МИР» 129820, ГСП, Москва, И-110, 1-й Рижский пер., 2

Ленинградская типография № 2 головное предприятие ордена Трудового Красного Знамени Ленинградского объединения «Техническая книга» им. Евгении Соколовой Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. 198052, г. Ленинград, Л-52, Измайловский проспект, 29.

© 1983 by Reston Publishing Company, Inc. A Prentice-Hall Company
© перевод на русский язык, «Мир», 1987

ПРЕДИСЛОВИЕ РЕДАКТОРА ПЕРЕВОДА

Важнейшими требованиями, предъявляемыми к системам памяти современных ЭВМ, являются повышение быстродействия, увеличение емкости и надежности. Эти противоречащие друг другу требования удается удовлетворить путем применения иерархических систем памяти с использованием различных типов запоминающих устройств (ЗУ) на разных уровнях. Подобный подход позволяет создать систему памяти, быстродействие которой приближается к быстродействию ее верхнего уровня, емкость — к суммарной емкости ЗУ всех уровней, а стоимость — к удельной стоимости ЗУ самого нижнего уровня.

Предлагаемая вниманию читателя монография посвящена вопросам организации работы и методам проектирования многоуровневой памяти и особенно ее верхнего уровня — кэш-памяти. По существу это первая работа, публикуемая на русском языке, в которой наиболее полно представлены проблемы, возникающие на этапах проектирования систем быстродействующей памяти и оценки ее основных характеристик.

В первых трех главах описано развитие систем памяти. Рассмотрены конкретные структуры систем памяти, используемых в широкоизвестных ЭВМ последних выпусков. Очень важно, что авторы дают конкретные рекомендации по проектированию систем с кэш-памятью и предлагают методы расчета эффективности процедур обмена информацией в многоуровневой памяти, в том числе с учетом стоимостных характеристик.

Отдельная глава посвящена использованию корректирующих кодов в быстродействующих системах памяти и традиционного метода оценки надежности таких систем с помощью λ-характеристик.

Большой интерес представляет гл. 7, в которой рассмотрена эволюция иерархических систем памяти и систем с кэш-памятью, а также прослежено развитие различных подходов к построению структур многоуровневой памяти, организации процедур обмена и формированию физических адресов на примерах систем памяти в ЭВМ IBM 360/85, IBM 370/155, IBM 370/165, IBM 3033, PDP-11/70, PDP-11/60, VAX-11/780, MV/8000 и ряда микропроцессорных систем с использованием схем управления памятью: Z8001, Z8010, MC6829, NS16082.

Книга предназначена для научных работников и специалистов, занимающихся разработкой систем памяти ЭВМ, а также для аспирантов и студентов, изучающих вычислительную технику.

Д-р техн. наук *В. И. Зуев*

ПРЕДИСЛОВИЕ

Эта книга была задумана как учебное пособие для студентов старших курсов и аспирантов. Она может оказаться полезной и для специалистов по ЭВМ, занимающихся разработкой систем памяти.

Главное внимание в книге уделяется основным принципам и теоретическим аспектам построения систем памяти. Несмотря на быстрый прогресс технологической базы запоминающих устройств, принципы разработки проектов этих систем остаются в значительной степени неизменными. Поэтому в книге речь идет главным образом о фундаментальных основах и методах, которыми должен владеть проектировщик. Чтобы показать справедливость теоретических результатов и определить набор технических характеристик памяти, приводятся результаты исследований, в ходе которых осуществлялась экспериментальная отработка аналитических средств.

В гл. 1 дан краткий обзор развития систем памяти. В гл. 2 рассмотрен простой пример, иллюстрирующий построение систем с кэш-памятью.

В гл. 3—6 достаточно полно отражены вопросы теории и проектирования быстродействующих систем памяти с кэшем. Гл. 5 содержит краткое описание мультипроцессорной системы с иерархически организованной памятью. Кроме того, там рассмотрены принципы страничной организации памяти в связи с их ролью в иерархических кэш-структурах.

В заключительной главе более подробно прослежено развитие систем с кэш-памятью и показано место иерархически организованной памяти в различных вычислительных системах.

А. Пом
О. Агравал

Глава 1

Быстродействующие запоминающие устройства ЭВМ

1.1. ВВЕДЕНИЕ

В течение последних трех десятилетий вычислительные системы прошли путь развития от простых машин фон Неймана до сложнейших систем с виртуальной памятью и разделением времени. Произошли глубокие изменения как в организации процессоров и систем памяти, так и в требованиях к памяти по быстродействию и емкости. На рис. 1.1 показан рост быстродействия и емкости систем памяти в течение последних трех десятилетий. Постоянно продолжающееся развитие вычислительной техники привело к рождению четырех поколений ЭВМ, а в последнее десятилетие — микропроцессоров. Микропроцессоры, выполненные на одном кристалле, потребляют мощность в единицы милливатт, стоят от 10 до 1000 долл. и в то же время обладают такими же или даже лучшими эксплуатационными характеристиками по сравнению с большими машинами тридцатилетней давности.

Эти значительные изменения, достигнутые за такой короткий промежуток времени, наилучшим образом можно проиллюстрировать, сравнив одну из первых ЭВМ с современной моделью. Машина UNIVAC I, введенная в эксплуатацию в 1951 г., имела память в 1000 слов, каждое из которых состояло из 12 буквенно-цифровых символов и бита контроля четности. Общая емкость внутренней памяти равнялась 84 Кбит. Это меньше емкости памяти большинства современных микропроцессорных систем, которая достигает 16 Кбайт (128 Кбит), и значительно меньше емкостей памяти мини-ЭВМ и основной памяти сверхбольших универсальных систем, которые достигают соответственно 0,5—1 Мбайт и 4—8 Мбайт. В табл. 1.3 и 1.4 приведены данные по емкости памяти некоторых современных микроЭВМ, мини-ЭВМ и больших универсальных вычислительных систем. Интересно сравнить стоимость памяти ЭВМ UNIVAC I со стоимостью полупроводниковой памяти современных микропроцессоров и мини-ЭВМ. Этот показатель уменьшился почти на три порядка с 20 долл. до 2 центов на бит. Что касается больших ЭВМ, то, например, машина ILLIAC IV, выпущенная в 1970 г., имела основную память емкостью 33 Мбит с временем доступа 120 нс и временем цикла 240 нс. По сравнению с ЭВМ

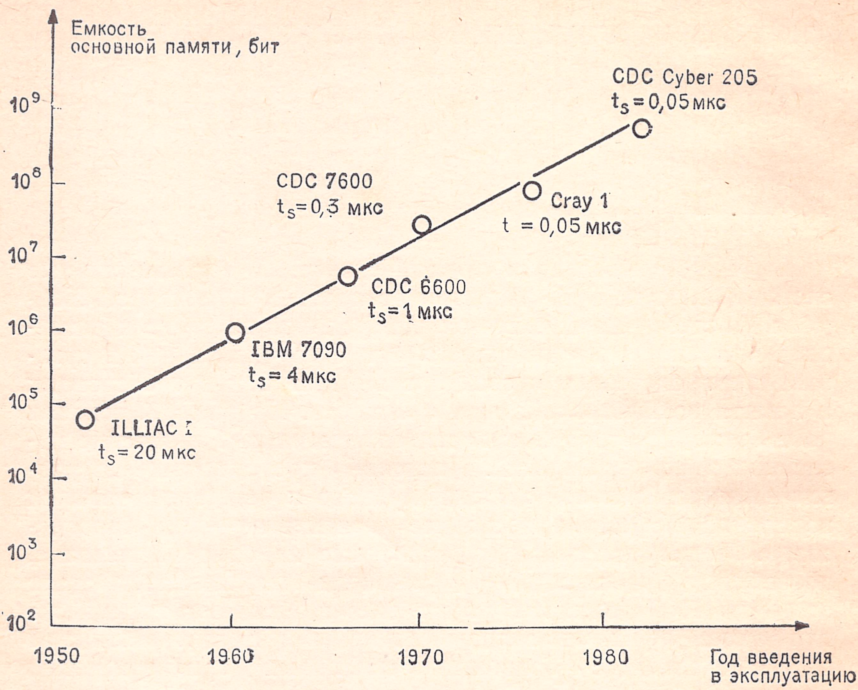


Рис. 1.1. Рост емкости основной памяти наиболее мощных ЭВМ (t_s — время цикла).

UNIVAC I емкость памяти увеличилась в 500, а быстродействие в 900—1000 раз.

Такое быстрое развитие систем памяти тесно связано с изменением производительности ЭВМ, которая в течение последних трех десятилетий каждые 5 лет в среднем возрастала в 10 раз.

Несмотря на эти улучшения параметров, очень часто возникало существенное несоответствие в быстродействии процессоров и устройств основной памяти ЭВМ. Из-за этого несоответствия производительность большинства вычислительных систем была ограничена быстродействием основной памяти. В результате для повышения производительности системы в целом необходимо было повышать быстродействие основной памяти.

Прогресс в области вычислительной техники, как правило, определяется принципом: «бóльшая производительность — за минимальную стоимость». Это приводит к тому, что для улучшения параметров вычислительных систем применяются новейшие достижения техники. По существу направления внедрения этих достижений можно разделить на две категории:

1) улучшение эксплуатационных характеристик ЭВМ за счет повышения быстродействия отдельных элементов;

2) улучшение эксплуатационных характеристик ЭВМ за счет обновления ее структуры.

Действительно, так как параметры элементов являются одним из ограничивающих факторов, производительность ЭВМ может быть повышена за счет использования наиболее быстродействующих элементов. Однако повышение быстродействия элементов ограничено технологическими возможностями и уровнем развития конструкторской базы. Кроме того, повышение быстродействия элементов не приводит к существенному возрастанию быстродействия ЭВМ в целом. Благодаря этому данный метод теряет свою привлекательность и заставляет искать другие пути повышения производительности вычислительных машин. Повышения быстродействия ЭВМ можно достичь также за счет уменьшения времени выполнения элементарных операций, таких, как сложение, умножение или деление. Однако исследования Винограда [V6] показывают, что в большинстве ЭВМ операция сложения выполняется в два, а операция умножения в три раза медленнее теоретически возможного предела. Таким образом, указанными выше методами не удается существенно повысить быстродействие вычислительных систем, что приводит к необходимости поиска новых архитектурных решений при разработке процессоров и памяти.

С точки зрения увеличения быстродействия и емкости памяти в этом случае представляют интерес следующие подходы: секционирование (расслоение) памяти, использование иерархии памяти и виртуальной памяти, разработка кристаллов памяти большего объема и быстродействия, применение кэш-памяти.

Ниже более полно рассматриваются методы построения кэш-устройств и иерархических структур памяти ЭВМ.

1.2. ОБЗОР ЗАПОМИНАЮЩИХ УСТРОЙСТВ И СПОСОБОВ ХРАНЕНИЯ ИНФОРМАЦИИ

Повышение производительности электронной вычислительной техники непосредственно связано с увеличением быстродействия и емкости памяти. За последние три десятилетия емкость памяти наиболее крупных вычислительных систем возросла от 1000 до 8 млн. слов, а время цикла уменьшилось с 20 мкс до 10 нс. Однако даже с учетом прогресса в технологии быстродействующие запоминающие устройства остаются более дорогими, чем медленные. Следовательно, с точки зрения стоимости эффективнее иметь иерархию памяти с небольшим по емкости запоминающим устройством, расположенным

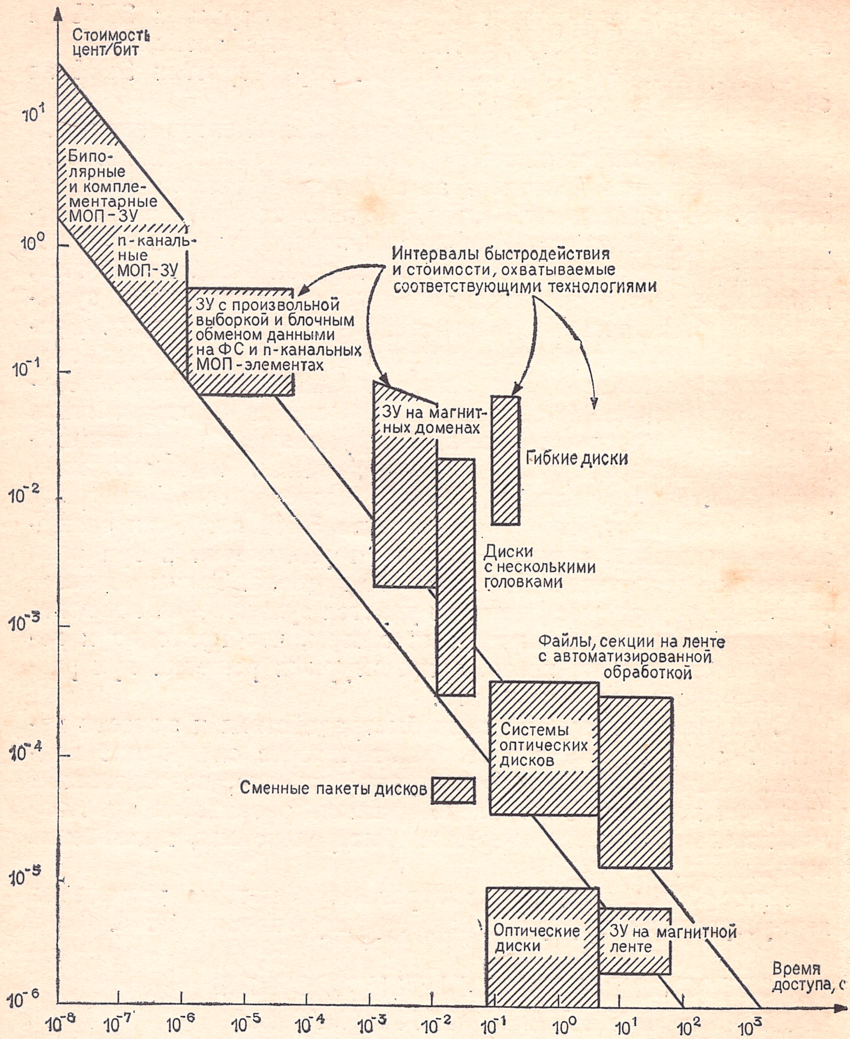


Рис. 1.2. Самоотношение стоимости и быстродействия различных устройств памяти. (Стоимость в среднем снижается на 25 % в год.)

рядом с процессором и имеющим минимальное время доступа. Такая иерархия позволяет согласовать характеристики памяти и центрального процессора. Рис. 1.2 иллюстрирует соотношение стоимости и быстродействия устройств памяти различных типов по данным на 1982 г.

По мере возрастания емкости памяти появилась необходимость использования больших модулей запоминающих

устройств. Это, к сожалению, привело к бóльшим физическим размерам устройств, увеличению длины проводников и задержки распространения сигналов. Хотя быстродействие и емкость отдельных устройств памяти возросли, этого было не достаточно для того, чтобы они соответствовали характеристикам процессора. В больших современных системах нередко время доступа процессора к памяти в два раза больше его цикла. Например, в IBM 360/95 память на плоских пленках емкостью 128 K 64-разрядных слов имеет время цикла 120 нс и время доступа 60 нс; фактическое же время доступа процессора к ней составляет 180 нс.

Использование в системе памяти должным образом выбранной иерархии является одним из методов удовлетворения противоречивых требований увеличения емкости памяти и быстродействия. При этом среднее время доступа ко всей памяти лишь незначительно превышает время доступа к небольшому быстродействующему запоминающему устройству, а количество хранящейся информации равно емкости большой памяти. Идея иерархии запоминающих устройств в том или ином виде с самого начала была одной из важных концепций при разработке вычислительных систем.

Иерархическая память — это система памяти, состоящая как минимум из двух запоминающих устройств, отличающихся быстродействием и емкостью. Обычно первое устройство памяти, расположенное в непосредственной близости от процессора, имеет малое время доступа и более высокую стоимость на бит. В связи с этим емкость более быстродействующей памяти в иерархии делается небольшой для того, чтобы оптимизировать стоимость всей системы. Технология памяти второго уровня иерархии выбирается исходя из низкой стоимости на один бит; такая память имеет бóльшие значения времени цикла и времени доступа.

Иерархическая память управляется пользовательскими программами, аппаратным путем или системным программным обеспечением так, чтобы система памяти по времени доступа приближалась к быстродействующей памяти, а по стоимости — к медленной памяти. Примером иерархии, управляемой пользователем, является оверлейная система на сердечниках и дисках; примером иерархии, управляемой системным программным обеспечением, является система со страничной организацией на сердечниках и дисках; типичным примером аппаратно-управляемой иерархии является система памяти, состоящая из кэш-памяти на биполярных схемах и устройствах на сердечниках.

В вычислительных системах всегда применялись те или иные формы иерархии запоминающих устройств. Даже простые калькуляторы имеют двухуровневую иерархию, состоя-

щую из внутренних регистров и внешнего пульта для ввода данных. Иерархия памяти существует благодаря разнице в стоимости устройств хранения информации. Если бы запоминающие устройства с минимальным временем доступа были достаточно дешевыми, то системы имели бы только один уровень памяти. Из-за отсутствия такой памяти требования высокой производительности и низкой стоимости наилучшим образом удовлетворяются использованием различных технологий, позволяющих сочетать дорогие быстродействующие устройства с дешевыми и медленными. Память, разрабатываемая на основе этой стратегии, называется иерархической системой памяти или многоуровневой системой памяти.

Иерархия памяти существует главным образом для того, чтобы повысить экономическую эффективность системы путем оптимального сочетания временных и стоимостных характеристик различных запоминающих устройств. Для того чтобы иерархия памяти была экономически эффективной, память должна иметь высокие эксплуатационные характеристики. Это означает, что технические параметры запоминающих устройств на разных уровнях иерархии должны существенно различаться. Невозможно достигнуть больших различий по эксплуатационным характеристикам и стоимости малой и большой памяти, если и та и другая выполнены по одной технологии и имеют близкие по значению времена доступа. В табл. 1.1 приведены эксплуатационные и стоимостные характеристики запоминающих устройств, выполненных по типичным современным технологиям, которые разделены на шесть категорий. Хотя данная таблица очень упрощена, она иллюстрирует тот факт, что все многообразие запоминающих устройств охватывается шестью типами как по стоимости, так и по эксплуатационным характеристикам.

Главной целью разработки эффективной иерархической системы памяти является обеспечение обмена необходимым количеством информации с процессором со скоростью наиболее быстродействующего запоминающего устройства, имеющегося в иерархии.

На ранней стадии развития вычислительной техники размер основной памяти, по современным представлениям, был достаточно мал и большая часть времени работы программы отводилась проблеме перекрытия, т. е. как и когда организовывать передачу информации между основной и вспомогательной памятью и включать для выполнения этих функций соответствующие команды в основную программу. Эти команды и процедуры, производящие разбиение программы на последовательность сегментов, которые перекрывали бы друг друга в памяти, были относительно простыми. Однако разработка эффективных методов перекрытия требовала от про-

Таблица 1.1. **Типы запоминающих устройств**

Тип	Время доступа и передачи	Снижение скорости	Емкость, бит	Стоимость [a], цент/бит	Основная технология
Регистры	2—20 нс	1	10^3—10^4	20—100	Полупроводниковые триггеры
Кэш-память	20—200 нс	10	10^4—10^5	1—10	Полупроводниковая память с произвольной выборкой
Основная память	200—2000 нс	100	10^6—10^8	0,1—0,5	Полупроводниковая память, ферритовые сердечники
Дополнительная память	1—10 мс	10^5—10^6	10^7—10^9	0,05—0,2	Диски и барабаны с фиксированными головками, цилиндрические магнитные домены
Вторичная память	20—100 мс	10^6—10^7	10^9—10^{11}	0,01—0,0005	Диски с подвижными головками
Массовая память	10—100 с	10^8—10^9	10^{11}—10^{12}	$(2—8) \times 10^{-5}$	Оптические диски, микрофильмы, магнитные ленты

а) Стоимость памяти уменьшается в среднем на 20—30 % в год. Данная таблица охватывает период 1980—1983 гг.

граммистов детального знания особенностей работы машины, на изучение которых они тратили больше времени, чем на решение собственных задач. Появление в середине 50-х годов языков высокого уровня значительно упростило задачу программистов. Эти языки дали им возможность достаточно просто строить большие программы и больше внимания уделять решению собственных задач, а не изучению особенностей работы машины. Однако с возрастанием сложности программ повышалась и роль их перекрытия. Фактически на решение этой проблемы затрачивалось от 25 до 40 % стоимости программирования. Было установлено, что плохие стратегии перекрытия могут существенно влиять на эффективность работы программы. С увеличением емкости запоминающих устройств эффективные стратегии перекрытия и иерархия памяти стали играть еще более важное значение. С тех пор как программисты перестали вникать в детали работы ЭВМ, их стало очень сложно убедить в необходимости разработки эффективных стратегий перекрытия. Это в свою очередь привело к необходимости увеличения емкости основной памяти.

Необходимость размещения больших программ в малых пространствах памяти стимулировала развитие аппаратных и программных средств ЭВМ с целью обеспечения *автоматического* перемещения информации между основной и вспомогательной памятью. Два решения этой проблемы привели к разработке принципов *статической* и *динамической* стратегий распределения памяти. Статический подход предусматривал наличие сведений о доступных ресурсах памяти и программе до начала ее работы, тогда как динамический подход не предполагал наличия предварительных сведений о программе и предусматривал сокращение или расширение выделяемой основной памяти в соответствии с текущими потребностями программы. Создание более совершенных, машинонезависимых, модульных, перемещаемых программ с эффективной структурой, использующей каталоги, делало статическую концепцию все менее и менее привлекательной.

Концепция динамического распределения памяти имела два направления развития. Одна школа предполагала использование динамического распределения памяти пользователем, другая отстаивала автоматическое распределение памяти системой. Сторонники динамического распределения памяти пользователем аргументировали свою позицию тем, что пользователи наилучшим образом осведомлены об алгоритмах работы своих программ и могут, таким образом, наилучшим образом осуществить динамическое распределение памяти. Их противники считали, что с увеличением емкости памяти пользователи будут не в состоянии эффективно и объективно решить эту проблему и задача распределения памяти в целом

должна быть передана системе. В 1961 г. группа специалистов в Манчестере (Англия) в рамках решения проблемы динамического распределения памяти выдвинула концепцию *виртуальной* памяти. Виртуальная память создавала у пользователя иллюзию одного большого, одноуровневого пространства хранения информации, все функции эффективного управления которым и перемещения информации между основным и вспомогательным запоминающими устройствами возлагались на систему. Таким образом, с точки зрения пользователя проблема распределения памяти была решена. Концепция виртуальной памяти, впервые реализованная в системе ATLAS, получила широкое признание и была использована в системах IBM 360/85, IBM 360/195, CDC 7600, B 6500 и более мощных сериях машин фирмы Burroughs, GE 645, IBM 370/158, IBM 370/168, SYMBOL-2R, PDP 11/45, PDP 11/70, а также во многих других.

Такие концепции, как поточная обработка данных, выглядевшая ранее скорее фантастической, чем реальной, мультипрограммирование, многопроцессорные системы, системы с разделением времени, которые получили развитие в сфере проектирования процессоров, а также расслоение памяти, иерархии памяти и виртуальная память, являются великолепными достижениями, позволившими улучшить общие эксплуатационные характеристики вычислительных систем. Однако все эти достижения дали возможность лишь частично устранить диспропорцию по быстродействию между процессором и основной памятью.

Идеально было бы построить основную память по той же технологии, что и регистры процессора, что позволило бы полностью устранить различие в их быстродействии. Но иметь большую основную память с временем цикла, соизмеримым с быстродействием процессора, экономически не выгодно. Примеры ранних разработок иерархической системы многоуровневой памяти, содержащих устройства с быстродействием в пределах от микросекунд до миллисекунд, указывали на возможность создания иерархии, включающей устройства, работающие в наносекундном и микросекундном диапазонах. Решение этой проблемы возможно с помощью кэш-памяти.

В системах с кэш-памятью быстрая и небольшая память, известная как буфер, или кэш, имеющая почти такое же быстродействие, как и регистры, расположена между процессором и основной памятью. Такой кэш согласует скорости процессора и основной памяти. Этот буфер совершенно незаметен, недоступен пользователям и не может быть непосредственно адресован. Однако использование кэша создает иллюзию, что большая основная память работает со скоростью процессора.

1.3. ПРОШЛОЕ И НАСТОЯЩЕЕ ИЕРАРХИЧЕСКОЙ БЫСТРОДЕЙСТВУЮЩЕЙ ПАМЯТИ

Концепция иерархии ЗУ наносекундного и микросекундного диапазонов (кэш-памяти) была выдвинута более десяти лет назад, но реализовать ее не представлялось возможным из-за соответствующего тому времени уровня развития технологии. Действительно, еще в 1962 г. Блум, Кохен и Портер [B1] предложили техническое решение, известное как «предварительно просматриваемая память», и показали, что, используя ассоциативную память на 256 слов с временем цикла 100 нс, можно получить эффективное время цикла системы памяти около 350—400 нс; в этой системе основная память имеет время цикла 1 мкс. [L1] Концепция нано- и микросекундной иерархии была также в элементарной форме реализована в других ЭВМ, таких, как Ferranti Atlas [B8] и ETL Mk-6 [T1]. В ЭВМ Atlas дополнительно к основной памяти на ферритовых сердечниках использовалась память совершенно нового типа емкостью в несколько тысяч слов, доступ к которой был чрезвычайно быстрым (0,2 мкс по сравнению с 0,75 мкс для памяти на ферритовых сердечниках). Эта память состояла из обмотки с небольшими ферритовыми стержнями. При этом содержимое ячейки памяти определялось наличием или отсутствием стержней. Однако такая память по существу применялась как постоянное запоминающее устройство для хранения подпрограмм и большого числа аналитических функций.

В ЭВМ ETL-Mk-6 использована трехуровневая иерархическая память, состоящая из запоминающих устройств на магнитном барабане, ферритовых сердечниках и туннельных диодах, с циклом 250 нс. Наиболее быстродействующая память была разделена на программный стек, арифметический стек и индексные регистры, причем программисту были доступны только последние две области памяти. Идея программного стека по существу заключалась в выделении определенного подпространства памяти для коротких подпрограмм. В 1965 г. Уилкс предложил использовать быстродействующую память на сердечниках, работающую как вспомогательное устройство для медленной памяти на сердечниках таким образом, что эффективное время доступа к системе приближается к времени доступа к быстрой памяти.

Все эти технические решения были реализованы в элементарной форме только в некоторых ЭВМ. В больших вычислительных системах из-за отсутствия подходящей технологии они использованы не были. Впервые нано- и микросекундная иерархия памяти была реализована с появлением полупро-

Таблица 1.2. **Различия между системой виртуальной памяти с кэшем и системой виртуальной памяти со страничной организацией**

Система виртуальной памяти с кэшем	Система виртуальной памяти со страничной организацией
Данные между основной памятью и кэшем передаются блоками по 4, 8, 16 или 32 слова	Данные между основной памятью и дополнительной памятью передаются страницами; размер страниц — 64 или 128 слов
Емкость кэша относительно мала (от 4 до 16 К байт)	Емкость основной памяти относительно большая
Разница в быстродействии кэша и основной памяти относительно мала (10—20 раз)	Разница в быстродействии основной памяти и вспомогательной памяти велика (1000—10 000 раз)
По быстродействию процессор и кэш практически не различаются	По быстродействию процессор и основная память существенно различаются
Так как процессор и кэш имеют одинаковое быстродействие, нет необходимости переключать центральный процессор на обработку другого задания на время передачи данных из основной памяти в буферную	Так как существует большая разница в быстродействии центрального процессора и основной памяти, для эффективного использования процессора целесообразно его переключение на выполнение другого задания на время передачи данных из вспомогательной памяти в основную и обратно
Из-за отсутствия диспропорции по быстродействию между процессором и кэшем алгоритм перемещения не является критичным для системы с кэш-памятью	Из-за разницы в быстродействии процессора и основной памяти алгоритм перемещения является чрезвычайно важным для основной памяти. Очень плохая стратегия перемещения информации может легко свести на нет достоинства всей системы
Из-за отсутствия разницы в быстродействии процессора и кэша существует широкий выбор алгоритмов записи; эксплуатационные характеристики системы памяти очень чувствительны к различным алгоритмам записи	Из-за разницы в быстродействии процессора и основной памяти выбор алгоритмов записи ограничен

водниковых интегральных запоминающих устройств в 1968 г. в большой вычислительной системе IBM 360/85.

Система с кэш-памятью имеет аналогию с другими системами, использующими иерархию памяти, например с такой, как система виртуальной памяти со страничной организацией. Целью создания системы виртуальной памяти со страничной организацией является достижение быстродействия основной памяти при стоимости, близкой к стоимости вспомогательной, или дополнительной, памяти, тогда как целью создания системы виртуальной памяти с кэшем является достижение быстродействия кэш-памяти при стоимости, близкой к стоимости основной памяти. Система виртуальной памяти с кэшем создает впечатление очень большого запоминающего устройства, работающего со скоростью кэша, или быстрого буфера. В системе виртуальной памяти внутренний уровень, или основная память, служит буфером для следующего уровня. Если между процессором и основной памятью находится небольшое быстродействующее запоминающее устройство, то такая система называется *системой виртуальной памяти с кэшем или системой виртуальной памяти с буфером.*

Проектирование системы виртуальной памяти с буфером аналогично проектированию системы виртуальной памяти со страничной организацией. Однако эти две системы различаются главным образом соотношением быстродействий двух внутренних уровней памяти. В табл. 1.2 указаны главные различия между системой виртуальной памяти с кэшем и системой виртуальной памяти со страничной организацией.

Кэш (или буфер) обеспечивает обмен информацией между памятью и процессором с более высокой скоростью и меньшими затратами по сравнению с отдельной быстродействующей большой памятью, а также создает впечатление, что большая память работает со скоростью кэша. До тех пор пока для современной памяти будут сохраняться различия в быстродействии и стоимости, подход с использованием буферной памяти будет оставаться экономически эффективным решением проблемы согласования скоростей процессора и основной памяти.

IBM 360/85 была первой большой вычислительной системой, в которой применен большой кэш для данных. Затем различные типы кэшей (адресный, программный, именной) использовались во многих больших ЭВМ. В табл. 1.3 приведены параметры кэш-памяти для некоторых из них.

В настоящее время кэш-память применяется почти во всех системах мини-ЭВМ. Впервые она была использована в мини-ЭВМ PDP 11/70 и PDP 11/60. Затем различные типы кэш-памяти появились в системах 32-разрядных мини-ЭВМ. Параметры некоторых из них приведены в табл. 1.4.

Таблица 1.3. **Большие вычислительные системы с кэш-памятью**

Модель	Производительность по отношению к IBM	Дата объявления фирмой	Время цикла процессора, нс	Емкость основной памяти, Мбайт	Время цикла основной памяти, нс	Расслоение	Емкость кэша, Кбайт	Время цикла кэша, нс
			Серии Amdahl 470V					
470V/5	3 × 158	02.77	32,5	8	320	4	16	65
470V/5-II	3 × 3031	10.78	32,5	8	320	4	32	65
470V/6	1,5 × 168	11.74	32,5	8	320	4	16	65
470V/6-II	1,5 × 3032	02.77	32,5	8	320	4	32	65
470V/7	1,2 × 3033	02.77	29	16	290	16	32	58
470V/7A	1 × 3033	08.79	29	16	290	16	32	—
470V/B	1,5 × 3033	10.79	26	16	260	16	64	52
			Фирма National Semiconductor					
AS/6	1,2 × 3032	10.77	72	16	360	4	64	72
AS/6-2	1,3 × 3032	10.77	72	16	360	4	64	72
AS/7-7033	1,1 × 3033	07.79	72	2—16	150	8	64	72
AS/8-7034	—	07.79	42	—	100	Нет	64	21
AS/3-3	1,6 × 138	09.78	115	2	1035 (чтение)	Нет	4	115
AS/3-4	1 × 148	09.78	115	4	1035 (чтение)	Нет	4	115
AS/3-5	—	02.79	115	8	920 (чтение)	—	4	115
AS/5-1	1 × 158	10.76	115	8	1035 690 (запись)	Нет	8	115
AS/5-3	1 × 158—3	10.76	115	8	920 (чтение) 690 (запись)	Нет	16	115
AS/6-2AP	2 × 3032	03.79	72	16	360	4	64	72
AS/7031	1 × 3031	12.78	92	8	460	Нет	32	92
AS/3000N	1,22 × 4341	01.80	115	2—4	920 (чтение) 690 (запись)	«	16	115
AS/3000	1 × 158—3	01.80	115	2—8	920 (чтение) 690 (запись)	«	8	115

Продолжение табл. 1.3

Модель	Производительность по отношению к IBM	Дата объявления фирмой	Время цикла процессора, нс	Емкость основной памяти, Мбайт	Время цикла основной памяти, нс	Расслоение	Емкость кэша, Кбайт	Время цикла кэша, нс
AS/7000N	1,4 × 3031	01.80	72	2—8	—	4	16	72
AS/7000	1,0—1,1×3033N	01.80	72	4—16	—	8	64	72
AS/7000DPC	—	01.80	72	4—16	—	32	64	21

Серии Sperry UNIVAC 1100

Модель	Наличие микропрограмм	Количество процессоров	Емкость основной памяти, Мбайт	Длина слова, бит	Время цикла, нс	Технология	Емкость кэша, Кбайт	Время цикла кэша, нс
1100/60, H1, H2, ECL	Да	1,2	0,5—1	36	575	МОП	8—16	116
1100/80, 80, 81, ECL	«	1	0,5—1	36	650	«	4—8	125
1100/80, 82, 83, 84, ECL	«	2, 3, 4	0,5—4	36	650	«	16—32	125

Серии универсальных ЭВМ Fujitsu FACOM M

Модель	Производительность	Дата объявления фирмой	Количество процессоров	Емкость основной памяти, Мбайт	Время цикла основной памяти, нс	Расслоение	Емкость кэша, Кбайт	Время цикла кэша, нс
FACOM M-180, II, IIAD	1 × 158-3	05.75	1—2	0,5—8	350	2—4	8—16	70
FACOM M-190	1 × 168-3	11.74	1—2	1—16	480	2—4	16	30
FACOM M-200	1 × 3031	01.78	1—4	2—16	—	2—4	64	—

Основные серии фирмы Hitachi

Модель	Количество процессоров	Емкость основной памяти, Мбайт	Время цикла основной памяти, нс	Расслоение	Емкость кэша, Кбайт
HITAC M-160H	1	2—6	—	Нет	8
HITAC M-170	1	0,5—8	—	2	32
HITAC M-180	1—2	1—16	—	4	64
HITAC M-200H	1—4	4—16	—	8	64

Таблица 1.4. **Системы мини-ЭВМ с кэш-памятью**

Система	Емкость кэша	Адресный кэш	Программный кэш	Именной кэш	Логически адресуемое пространство
VAX 11/780	8 Кбайт	—	8 Кбайт	Нет	4 Гбайт
IBM 4331	8 Кбайт	—	—	«	16 Мбайт
PRIME 750	16 Кбайт	—	1 блок	«	512 Мбайт
PE 3240	8 Кбайт	—	16 байт	«	16 Мбайт
SEL 32/77	4 байт	—	На одну команду	«	16 Мбайт
MV/8000	16 Кбайт	—	1 Кбайт	«	4 Гбайт

Кэш-память начинает применяться почти везде. Трудно представить себе какую-либо вычислительную систему, по крайней мере следующего десятилетия, без различных типов кэшей. В настоящее время они нашли применение в микроЭВМ и системах с очень большой степенью интеграции. В перспективе кристаллы микроЭВМ очень большой степени интеграции будут содержать различные типы кэшей. Более подробно история развития систем виртуальной памяти и систем с кэш-памятью излагается в гл. 7.

УПРАЖНЕНИЯ

1. Отсутствие более дешевых, быстродействующих и бóльшей емкости кристаллов памяти привело к различным новым решениям в структуре процессор — память. Проанализируйте, к чему может привести появление очень быстродействующей, бóльшей по объему и более дешевой системы основной памяти. Какое влияние это может оказать на организацию программного обеспечения? Будет ли использоваться мультипрограммирование? Что изменится в схеме виртуальной памяти? Что произойдет с концепциями кэша? Будут ли вычислительные системы будущего иметь кэши?

2. Какие факторы должны учитываться при проектировании и построении иерархической системы памяти?

Глава 2

Основы построения буферной памяти (кэш-памяти)

2.1. ВВЕДЕНИЕ

Несмотря на быстрое развитие технологии запоминающих устройств, во многих современных системах стоимость памяти является одной из основных характеристик аппаратной части ЭВМ. Как отмечено в гл. 1, буферная (или кэш) организация памяти является экономически эффективной даже в устройствах с относительно небольшой емкостью для достижения высокого быстродействия при незначительном возрастании стоимости хранения одного бита информации по сравнению с медленной памятью. Концепция кэш-памяти нашла практическое применение благодаря двум аспектам: программному и экономическому. Программный аспект заключается в том, что большинство стандартных пользовательских программ создается в соответствии с «принципом районирования», который означает, что в любой момент времени адресуемая часть программы размещается в некоторой ограниченной части своего адресного пространства. Экономический аспект обусловлен отсутствием более дешевой по сравнению с кэшем технологии, позволяющей достичь аналогичных эксплуатационных характеристик. Кэш обеспечивает обмен памяти с процессором небольшим количеством информации с большой скоростью и тем самым создает впечатление, что большая основная память работает со скоростью кэша. До тех пор пока элементная база медленной памяти будет значительно дешевле, чем быстрой, кэш-память будет оставаться экономически эффективной.

В настоящей главе будут описаны основные элементы кэш-памяти и приведены рекомендации по проектированию и анализу конкретного варианта кэша, иллюстрирующие, как, используя небольшую кэш-память, можно существенно улучшить эксплуатационные характеристики запоминающих устройств при небольшом возрастании их стоимости. Сначала будет рассмотрен простой метод проектирования и оценены различные параметры для иллюстрации этого метода. Более сложному методу проектирования и анализу различных аспектов буферных запоминающих устройств посвящены последующие главы.

2.2. ОСНОВНЫЕ ЭЛЕМЕНТЫ КЭШ-ПАМЯТИ

Идея, лежащая в основе создания буфера (кэша), заключается в использовании быстрого и относительно небольшого запоминающего устройства между процессором и основной памятью. Кэш служит звеном, согласующим скорости процессора и памяти, при этом все данные в процессор поступают из кэша.

1. Буферная память (или кэш-память) по существу состоит из небольшого быстродействующего запоминающего устройства, содержащего часть информации основной памяти. Такой информацией могут быть адреса, команды или данные. В соответствии с этим кэш может быть адресным, программным или кэшем данных. Быстродействие этого небольшого запоминающего устройства обычно на порядок выше, а емкость на один-два порядка меньше, чем у основной памяти.

2. В системе с кэш-памятью требуется *идентификатор*, или «признаковая память», для того, чтобы определить, какие части информации основной памяти были скопированы в кэш-память. В зависимости от организации обмена эти части могут быть словами, блоками или страницами. Специальная память, содержащая имена частей или адреса, обычно называется *признаковой памятью*, или *справочником*, и является ассоциативной.

3. В буферной памяти должна быть логическая схема, определяющая порядок удаления слов или блоков из кэша при необходимости записи в него новой информации из основной памяти. Такая логическая схема называется *приоритетным списком обновления данных*.

4. Для синхронизации различных действий, например обращения в признаковую память, выборки и перемещения данных, в систему кэш-памяти вводится устройство управления.

Перечисленные требования в зависимости от конкретных условий проектирования и стоимости могут быть реализованы различными способами. Для наглядности рассмотрим структуру буферной памяти, представленную на рис. 2.1. Факторы, влияющие на размеры буфера, конкретные варианты его построения, схемные соображения, а также списки обновления данных будут рассмотрены в последующих главах.

2.3. ЭКОНОМИЧЕСКАЯ И ЭКСПЛУАТАЦИОННАЯ ЭФФЕКТИВНОСТЬ

На примере рис. 2.1 можно показать, почему буферная память является экономически выгодной. Для оценок мы будем использовать представленные фирмами-изготовителями

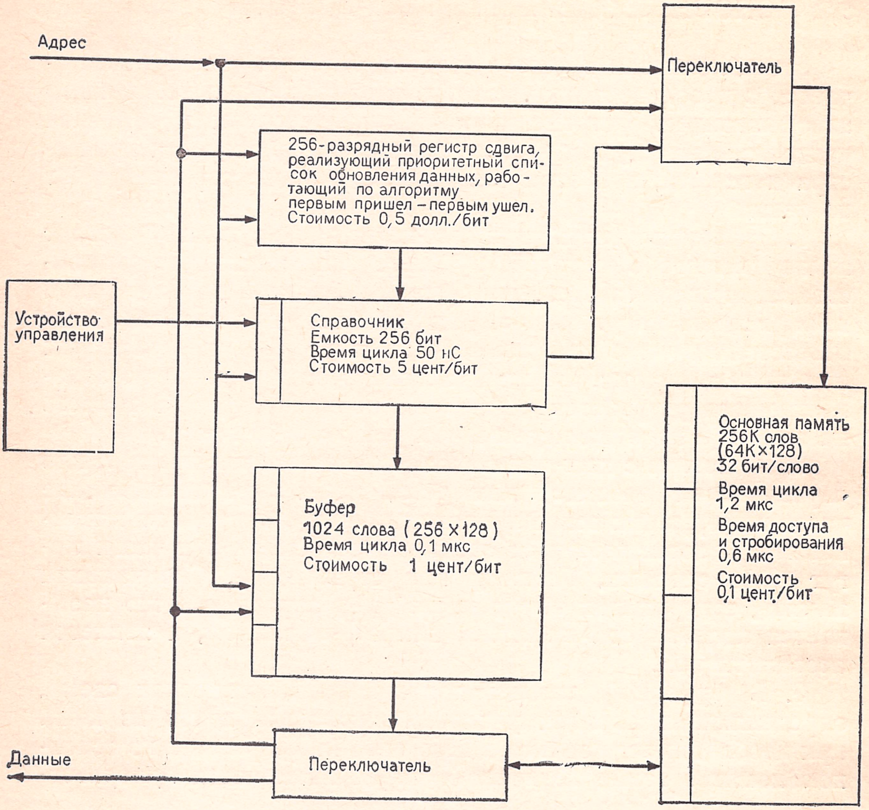


Рис. 2.1. Пример буферной памяти.

данные на 1981—1983 гг. о стоимости и размерах оперативных и буферных запоминающих устройств, которые могут быть использованы в подобной системе. Следует иметь в виду, что стоимость запоминающих устройств со временем уменьшается, но соотношение стоимостей различных типов памяти почти не меняется. В данном примере рассматривается кэш-память. Информация между кэшем и основной памятью передается блоками по 16 байт, а кэш и основная память организованы так, что в них имеются сверхдлинные слова в 16 байт. В этом случае обмен сверхдлинным машинным словом с основной памятью можно произвести за один цикл. В рассматриваемом примере емкость основной памяти равна 1 млн. байт (1 Мбайт) и адресация осуществляется 20-разрядным адресом. Способ адресации основной памяти иллюстрируется рис. 2.2. В данном случае мы имеем пример машины с адресуемыми байтами. Четыре младших

разряда адреса используются для адресации байта внутри каждого из 64 тыс. блоков, состоящих из 16 байт. В данном примере буфер содержит 256 блоков, каждый из которых включает 16 байт. Буфер организован как единое, полностью ассоциативное запоминающее устройство. Так как буфер имеет пространство для 256 блоков, признаковая память также имеет пространство для 256 элементов, каждый из которых соответствует блоку в 16 байт. Приоритетный список обновления данных организован как простой 256-разрядный регистр сдвига, один бит которого соответствует блоку информации в буфере.

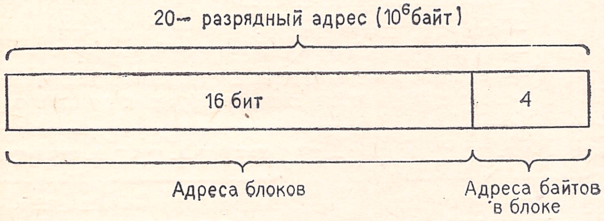


Рис. 2.2. Адресный формат.

Для сравнения стоимости буферной системы памяти с системой, не содержащей буфер, рассчитаем оптимальные эксплуатационные характеристики буферной системы памяти, а затем определим ее стоимость. Далее сравним полученные стоимость и эксплуатационные характеристики с параметрами системы памяти, состоящей из быстродействующих запоминающих элементов одного типа. Для того чтобы оценить эффективные характеристики системы памяти и максимально упростить анализ при расчете быстродействия буферной памяти, допустим, что для передачи информации в буфер из основной памяти, когда необходимо загрузить новый блок и удалить старый, используется простой алгоритм обмена. Таким образом, когда новое слово (или блок) должно быть передано из основной памяти в кэш, необходимое для него в кэше пространство освобождается за счет передачи старого слова (или блока) в основную память. Далее предполагается, что система памяти будет занята до тех пор, пока не завершится операция пересылки информации из одного запоминающего устройства в другое.

Эффективное время цикла буферной системы памяти может быть рассчитано с помощью параметров системы и вероятности нахождения требуемого слова в буфере. Вероятность нахождения требуемых данных в буфере (вероятность удачного обращения) определяется как отношение числа запросов, при которых требуемые данные находились в буфере,

к общему числу запросов. Характеристики системы могут быть вычислены с помощью следующих параметров: времени поиска указателя, вероятности удачного обращения, времени цикла кэш-памяти, или буферной памяти, времени цикла медленной памяти.

Эти параметры определены в табл. 2.1.

Таблица 2.1

TEFF	— эффективное время цикла для системы
TFRC	— время цикла чтения из буфера
TFWC	— время цикла записи в буфер
TSRH	— время поиска указателя
TSRC	— время цикла чтения из основной памяти
TSWC	— время цикла записи в основную память
HR	— вероятность нахождения требуемых данных в буфере
MR(1 — HR)	— вероятность отсутствия требуемых данных в буфере

$$\mathrm{TEFF} = \underbrace{\mathrm{TSRH} + \mathrm{HR}\ \mathrm{TFC}}_{\text{Выборка из буфера}} + (1-\mathrm{HR})\underbrace{(\mathrm{TSRH} + \mathrm{TFC} + \mathrm{TSRC} + \mathrm{TSWC}}_{\text{Обращение к дополнительной памяти}}$$

Значения параметров

$\mathrm{HR} = 0{,}95$ $\qquad \mathrm{TFC} = \mathrm{TFRC} = \mathrm{TFWC} = 100$ нс

$\mathrm{TSRH} = 50$ нс $\qquad \mathrm{TSRC} = \mathrm{TSWC} = 1200$ нс

$\mathrm{TEFF} = 50 + 0{,}95 \times 100 + 0{,}05\ (50 + 100 + 1200 + 1200) = 275$ нс

Используя значения быстродействия, приведенные в табл. 2.1, и положив вероятность удачного обращения равной 0,95, по данному в табл. 2.1 соотношению рассчитаем эффективное время цикла буферной системы памяти. Одним из наиболее критичных параметров системы является вероятность удачного обращения. В данном случае используется наиболее типичное значение этого параметра, а более детальное экспериментальное обоснование будет приведено позже. Подставив соответствующие величины, получим для данной системы памяти эффективное время цикла, равное 275 нс.

В табл. 2.2 приведен расчет стоимости системы на основе указанных на рис. 2.1 цен на один бит, включающей стоимости устройств, реализующих функции управления. Как видно из табл. 2.2, стоимость буферной системы лишь на 9,3 % больше стоимости одной медленной памяти. Быстродействие же возрастает более чем в 4 раза (275 нс/1200 нс).

Для приблизительных вычислений можно принять, что стоимость большой памяти пропорциональна квадратному корню скорости. Таким образом, если бы запоминающее устройство с эффективным временем цикла 275 нс было построено без буфера, а с помощью быстродействующих элемен-

Таблица 2.2

А. Буферная система	
Устройство управления	100,00
Переключатели	100,00
Буфер: 32 768 × 0,01	327,70
Указатель: 4096 × 0,05	204,80
Таблица обновления данных: 256 × 0,5	128,00
Основная память: 8 388 608 × 0,001	8388,60
	9249,10 долл.
Увеличение стоимости	860,50 (9,3 %)
Повышение быстродействия	$\frac{1200}{275} = 4{,}36$
Б. Система без буфера с быстрой основной памятью	
$8\,388\,608 \times \sqrt{4{,}36} \times 0{,}001 = 17\,516$ долл.	

тов памяти, его стоимость возросла бы в два раза по сравнению со стоимостью основной памяти. В части Б табл. 2.2 показан расчет стоимости такого быстродействующего запоминающего устройства.

Отметим, что используя небольшой быстродействующий буфер (приблизительно в 250 раз меньший по емкости и с быстродействием в 12 раз выше по сравнению с основной памятью) и учитывая, что вероятность нахождения требуемых данных в буфере большая, можно создать буферную систему памяти с высокими эксплуатационными характеристиками и меньшей стоимости по сравнению со стоимостью отдельного большого быстродействующего запоминающего устройства.

В последующих главах будут подробно исследованы различные параметры проектирования, а также методики и алгоритмы, применяемые в системах с кэш-памятью.

УПРАЖНЕНИЯ

1. Используя пример системы памяти, приведенный в главе, вычислите эффективное время цикла памяти для значений вероятности удачного обращения, изменяющихся от 0 до 1 с шагом 0,1. Постройте график зависимости эффективного времени цикла памяти от вероятности удачного обращения. Проанализируйте смысл значений вероятности удачного обращения, равных 0 и 1.

2. Какой должна быть вероятность удачного обращения, чтобы эффективное время цикла системы памяти было таким же, как у медленной основной памяти? Какова будет стоимость при этом значении вероятности? Проанализируйте преимущества и недостатки такой организации.

3. В предположении заданных стоимостей буфера и других электронных устройств определите емкость основной памяти, при которой средние стоимости на бит для системы памяти и отдельного быстродействующего запоминающего устройства будут равны ($4{,}36 \times 0{,}1$ цент/бит, HR = 0,95).

4. Определите, как снизится быстродействие системы, если для передачи одного блока информации из основной памяти потребуется не один, а четыре цикла чтения.

5. Каково будет эффективное быстродействие системы памяти при вероятности удачного обращения, равной 0,9, в случае использования простого алгоритма сквозной записи (т. е. когда не требуется обратной передачи данных из буфера в основную память)? Положите соотношение операций чтения и записи равным 4 : 1. Пусть в обоих случаях значение вероятности удачного обращения будет одинаковым. Обсудите такую систему.

Глава 3

Проектирование систем с кэш-памятью

3.1. ВВЕДЕНИЕ

В этой главе приводятся соображения по проектированию систем виртуальной памяти с кэшем. Анализируются различные факторы, определяющие эффективное время цикла системы с кэш-памятью, а затем вычисляются значения времени цикла для больших и малых систем памяти. Сначала эти факторы анализируются применительно к малым вычислительным системам, основная память которых невелика по емкости и может содержать всего один-два модуля. Оптимальная организация буферов в таких системах отличается от организации аналогичных по назначению структур, которые используются в системах, содержащих множество модулей основной памяти, как это имеет место в больших вычислительных системах. Однако основные методы разработки одинаковы для систем памяти любых типов. Затем рассматриваются большие системы памяти. Основное внимание уделяется системам с двухуровневой иерархией.

Эффективный подход к проектированию систем виртуальной памяти с буфером включает следующие этапы.

- Разработку подходящей иерархической системы памяти (содержащей ЗУ, работающие в наносекундном и микросекундном диапазонах), имеющей минимальную стоимость и максимальную производительность.
- Проектирование системы с кэш-памятью, обеспечивающей работу с логическим, т. е. виртуальным пространством памяти, бо́льшим, чем пространство физических адресов. Такая система должна включать средства автоматического управления для перераспределения памяти, дающие пользователю возможность не учитывать размеры физической памяти.
- Выбор рациональной длины единицы обмена информацией между различными уровнями памяти и разработку эффективной схемы преобразования адресов. Эффективность схемы преобразования адресов определяется ее способностью осуществлять отображение адресов таким образом, чтобы минимизировать эффективное время обращения к памяти.

Формирование эффективной стратегии управления памятью включает разработку важных методов трех видов:

методов отображения, которые представляют собой ме-

тоды преобразования адресов и методы распределения адресного пространства;

методов выборки, т. е. алгоритмов свопинга;

методов или стратегий замещения информации.

Кэш-память должна быть небольшой и быстродействующей и в то же время обеспечивать высокую вероятность нахождения в ней нужных в данный момент информационных слов. Поскольку кэш-память должна быть небольшой по емкости (по сравнению с памятью более низкого уровня), необходим какой-либо механизм отображения адресов памяти более низкого уровня (т. е. основной памяти) в адреса кэш-памяти. Так как кэш-память должна быть быстродействующей, преобразование адресов не должно отнимать излишнего дополнительного времени. Таким образом, в идеале методы отображения адресов любой кэш-структуры должны быть простыми и недорогими в реализации.

Алгоритмы свопинга, т. е. методы выборки для основной и буферной памяти, должны обеспечивать либо получение всей информации по требованию (*правило выборки по запросам*), либо предварительное распределение информации (*правило выборки с упреждением*). Методы выборки, используемые на различных уровнях иерархии памяти, могут быть одинаковыми или разными. Занесение информации в кэш-память обычно производится по запросам.

Выбор эффективного метода или стратегии замещения подразумевает достижение максимальной частоты удачных обращений и повышение тем самым общей производительности системы.

Характеристики проектируемой системы памяти, снабженной средствами буферизации, можно разделить на две группы: внутренние и внешние. Эффективное быстродействие буфера и эффективная стоимость представляют собой внутренние характеристики. К числу внешних характеристик, которые устанавливаются опытным путем и используются при проектировании в качестве исходных данных, относятся вероятность удачного обращения, размещение буфера, емкость буфера, размер блока, алгоритмы управления, отображение адресов, отношение значений времени обращения для чтения и записи, алгоритмы записи, число модулей основной памяти и последовательность обращения к ним, эффективная задержка или время ожидания для одного модуля, эффективная средняя задержка или время ожидания для нескольких модулей.

Эффективное быстродействие буфера должно быть примерно равно быстродействию процессора. Если полное эффективное время обращения к системе памяти с использованием буфера превышает время базового цикла процессора,

то для уравнивания ее быстродействия с быстродействием процессора можно прибегнуть к конвейерной организации процесса обработки информации. Когда кэш-память работает с такой же скоростью, как процессор, никакой необходимости в конвейерной работе процессора нет. Делать быстродействие кэш-памяти более высоким, чем быстродействие процессора, не имеет смысла, так как процессор (или процессоры) требует обслуживания со стороны памяти с интервалом, не меньшим чем время его рабочего цикла.

Кроме того, при проектировании эффективной кэш-памяти следует принять во внимание следующие обстоятельства:

Проектируемая система имеет традиционную или нетрадиционную архитектуру?

Является она большой, средней или весьма малой системой?

По физическому или функциональному принципу должен быть размещен буфер?

Буфер следует организовать как однородное или разнородное устройство?

Память с буфером эффективнее функционирует потому, что обращения производятся по ограниченному количеству адресов, т. е. некоторые слова, находящиеся в памяти, используются гораздо чаще, чем другие, и в результате реализации стратегии перемещения и алгоритмов свопинга выявляется тенденция к постоянному пребыванию этих более часто используемых слов в буфере. Количество обращений к кэш-памяти может быть определено с помощью вероятности удачного обращения. В одной из последующих глав будет показано, каким образом на вероятности удачного обращения сказываются такие факторы, как объем буфера, тип задачи, алгоритм свопинга, стратегия перемещения и размер блока. Однако в данной главе для изучения организации и функционирования памяти с буфером удобно использовать основные параметры. Поэтому системы памяти будут оцениваться главным образом с использованием вероятности удачного обращения, доли записываемых слов и слов с флагом, количества участвующих в обращении к памяти модулей, стратегии перемещения и типа используемых алгоритмов свопинга.

В зависимости от особенностей проектируемой системы разработчик может обнаружить, что один из нескольких известных алгоритмов свопинга оказывается наиболее эффективным. В данной главе будут рассмотрены различные варианты организации буферов, и построенные с ними системы будут сравниваться с точки зрения их производительности и других существенных характеристик. Наиболее важным критерием для сравнения систем памяти, снабженных средствами

буферизации, является эффективное время цикла памяти при заданной вероятности удачного обращения. Поэтому именно этот критерий и будет использоваться в качестве основного.

Одним из способов, часто используемых для повышения производительности памяти, является совмещение во времени операций, выполняемых с модулями основной памяти и с буфером. За счет того, что длительность операции ЦП включает время выборки из памяти, можно добиться повышения производительности системы; поэтому будет проведено сравнение систем, в которых совмещение используется и не используется. Системы памяти будут также изучены с точки зрения эффективности использования флага записи для слов, находящихся в буфере.

3.2. РАЗДЕЛЕНИЕ БУФЕРА

Память с буфером может быть организована множеством способов. Одним из важных аспектов ее организации является метод, посредством которого содержимое справочника или поисковой памяти указывает на информацию, находящуюся в буфере. Буфер может быть организован таким образом, что любой блок, т. е. группа слов, может поступить в него из любой части адресного пространства основной памяти; такая организация буфера требует построения его в виде ассоциативного ЗУ. Кроме того, как показал Гибсон [G1], память с буфером может быть разделена на несколько частей, каждая из которых представляет собой память с буфером, т. е. часть буфера обслуживает часть основной памяти. Память с такой организацией обычно называют многоклассовой; ее достоинством является относительная простота аппаратной реализации.

На рис. 3.1 иллюстрируется организация памяти, при которой буфер разделен на несколько классов. В данном случае информация пересылается в буфер и из него блоками по четыре слова; каждый из классов буфера обслуживается буферными сегментами. Эффективность такой организации зависит от того, каким образом выполняется присвоение адресов, и от характера процедуры сборки блока слов для размещения его в основную память. В системах, организованных подобным образом, каждый последующий блок слов становится частью нового класса. Так делается для того, чтобы при последовательном просмотре следующих друг за другом ячеек могло быть использовано множество классов буфера, в результате чего весь буфер при реализации конкретной задачи может оперативно работать.

Более наглядно многоклассовая память с буфером может быть описана с помощью следующих простых положений.

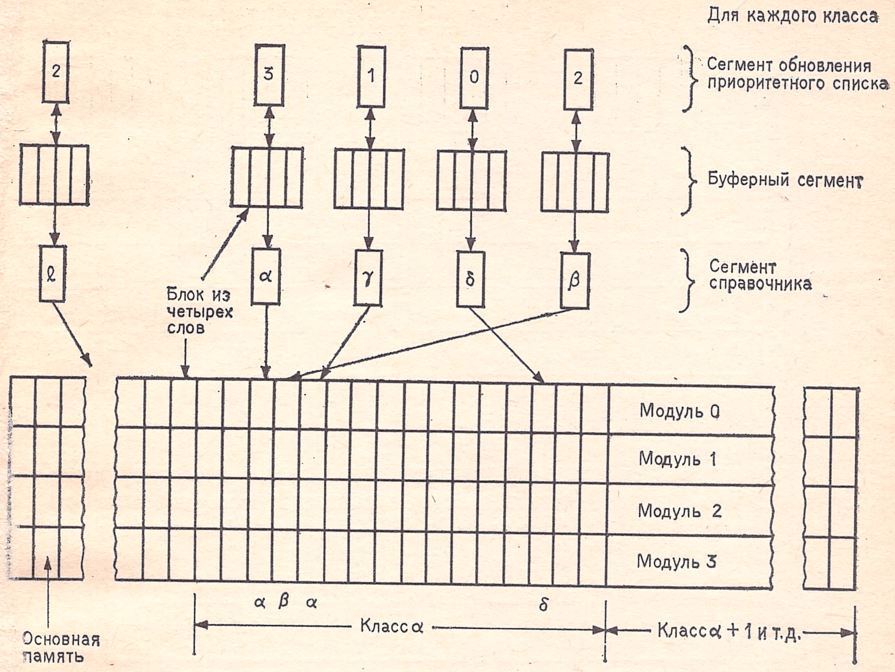


Рис. 3.1. Использование справочника в многоклассовой памяти.

Если буфер разбит на две части, каждая из которых обслуживает половину основной памяти, то образуется система, состоящая из двух классов. Если буфер разбит на 32 отдельные части, каждая из которых обслуживает 1/32 часть основной памяти, имеет место 32-классовый буфер. Исследование вероятности удачного обращения для традиционно организованных систем показало, что если присвоение адресов выполнено рациональным образом, то существенное увеличение числа классов может почти не сказываться на значении этого показателя. При этом снижается сложность аппаратурной реализации буфера. Таким образом, использовать многоклассовые буферы целесообразно особенно в небольших системах памяти.

Если число классов в буфере увеличивается в такой степени, что каждый буферный блок обслуживает конкретную часть основной памяти, то имеет место буферная память с *прямым отображением*. При этом значение вероятности неудачного обращения превышает величину, характерную для буферной памяти, построенной по ассоциативному принципу. Буфер с прямым отображением показан на рис. 3.2.

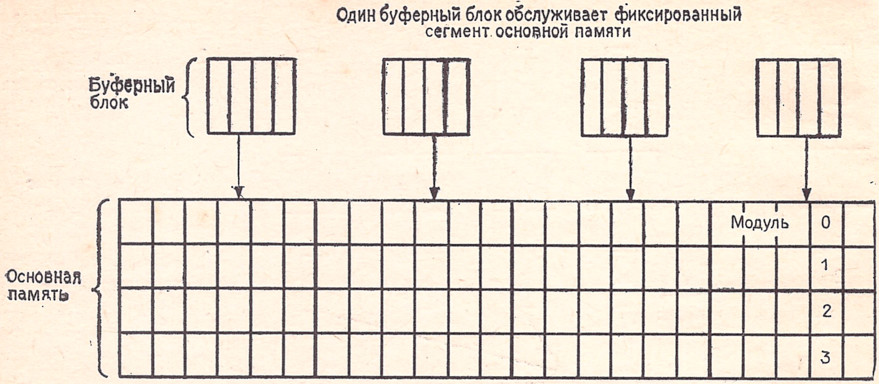


Рис. 3.2. Буфер с прямым отображением.

На рис. 3.3, *а* иллюстрируется присвоение адресов, которое обычно должно иметь место в многоклассовом справочнике системы, которая содержит 16-байтовые блоки, каждый

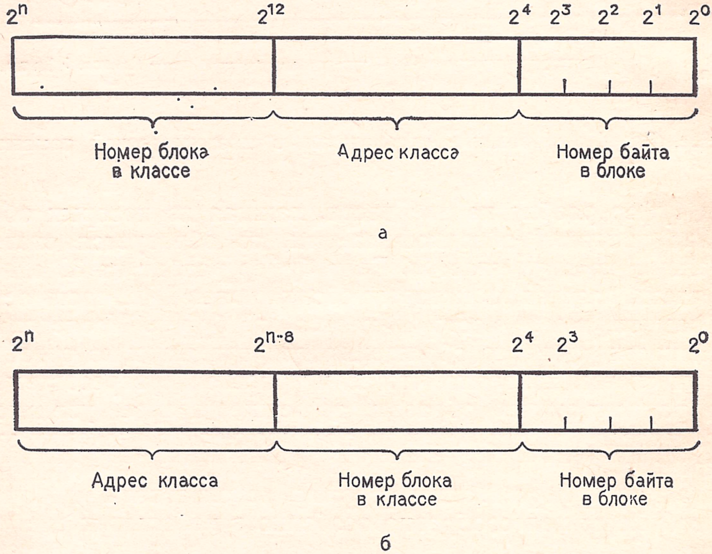


Рис. 3.3. «Нормальное» присваивание адреса содержимому буфера (*а*) и присваивание адреса с пониженной эффективностью (*б*).

из которых составляет класс; всего в буфере 256 классов. Адреса словам памяти обычно присваиваются последовательно в процессе сборки программных сегментов. Для того чтобы почти весь буфер участвовал в реализации задачи, к этому

должны быть привлечены все классы. На рис. 3.3, *б* иллюстрируется менее эффективный способ присвоения адресов. Аналогичные результаты были бы получены, если бы программа состояла из очень коротких сегментов по одному блоку.

3.3. АЛГОРИТМЫ СВОПИНГА

Алгоритм свопинга определяет порядок и способ пересылки данных между кэш-памятью и основной памятью. Задачей данного раздела этой главы является освещение наиболее характерных особенностей различных алгоритмов свопинга, применяемых в небольших системах памяти.

В таких системах основная память рассматривается как состоящая из одного модуля, хотя каждое слово памяти может содержать несколько слов, имеющих формат, присущий данной ЭВМ. Алгоритмы свопинга могут быть самыми разнообразными. Мы ограничимся рассмотрением четырех из них. Для некоторых алгоритмов свопинга существенными являются параметры, не используемые в случае простого свопинга. Одним из таких параметров является доля буферных блоков, в которые производится запись, после того, как они перемещены в основную память. Если в некоторый блок за время его пребывания в буфере запись не происходит, то нет необходимости перемещать его обратно в основную память, потому что там по-прежнему находится его достоверная копия. Далее вследствие того, что ЦП может продолжать обработку информации, пока завершается цикл основной памяти, возможна задержка. Она представляет собой просто среднее время, которое затрачивается на ожидание момента, когда из основной памяти в буфер можно переслать новый блок. Иногда оказывается, что основная память занята и буферу приходится ожидать в течение почти целого цикла основной памяти, пока он сможет обратиться к ней за операндом.

При всех последующих расчетах предполагается, что вероятность отсутствия блока в буфере характеризуется одним параметром — вероятностью неудачного обращения, причем вероятности отсутствия в буфере различных блоков статистически независимы [E2]. Конечно, во многих ситуациях это допущение несправедливо; однако подобные упрощения позволяют создавать эффективные аналитические средства расчета.

Если ЦП начинает обработку информации сразу же после получения слова из основной памяти, можно достичь некоторой экономии времени за счет того, что ЦП работает с буфером, пока завершается рабочий цикл модуля памяти. Сред-

нее время ожидания зависит не только от вероятности неудачного обращения, но и от соотношения чисел операций чтения и записи, а также от того, какая часть блоков данных отмечена флагом (подвергается записи за время нахождения в буфере). Обе эти величины полагаются статистически независимыми.

3.3.1. Типы алгоритмов свопинга

В системах памяти с буфером для обмена данными между буфером и основной памятью могут применяться алгоритмы записи или свопинга нескольких типов. Способ, которым записывается информация в системах с основной памятью (в частности, в основную память), является важным фактором при расчете эффективного времени цикла.

Применительно к разным системам памяти с буфером были исследованы следующие типы алгоритмов:

алгоритм сквозной записи (WT) или сквозного накопления (ST);

алгоритм простого свопинга (SS) или обратной записи в конфликтных ситуациях (CUW);

алгоритм свопинга с флагами (FS) или обратной записи в конфликтных ситуациях с флагами (CUX);

алгоритм регистрового свопинга с флагами (FRS).

Примечание: последующее описание алгоритмов свопинга применимо к пересылке как отдельных слов, так и блоков.

3.3.2. Алгоритм сквозной записи (WT — Write Through)

Это самый простой алгоритм записи из числа используемых в системах памяти с буфером. В соответствии с этим алгоритмом каждый раз при появлении запроса на запись по некоторому адресу обновляется содержимое области по этому адресу как в быстрой, так и в основной памяти, даже если копия содержимого по этому адресу находится в быстром буфере. Такое постоянное обновление содержимого основной памяти, как и буфера, при каждом запросе на запись позволяет постоянно поддерживать информацию, находящуюся в основной памяти, в обновленном состоянии. Поэтому, когда возникает запрос на запись по адресу, относящемуся к области, содержимое которой не находится в данный момент в быстром буфере, новая информация записывается просто на место слова (*блока*), которое предполагается переслать в основную память (без необходимости пересылки этого слова в основную память), так как в основной памяти уже находится его достоверная копия. Наряду с не-

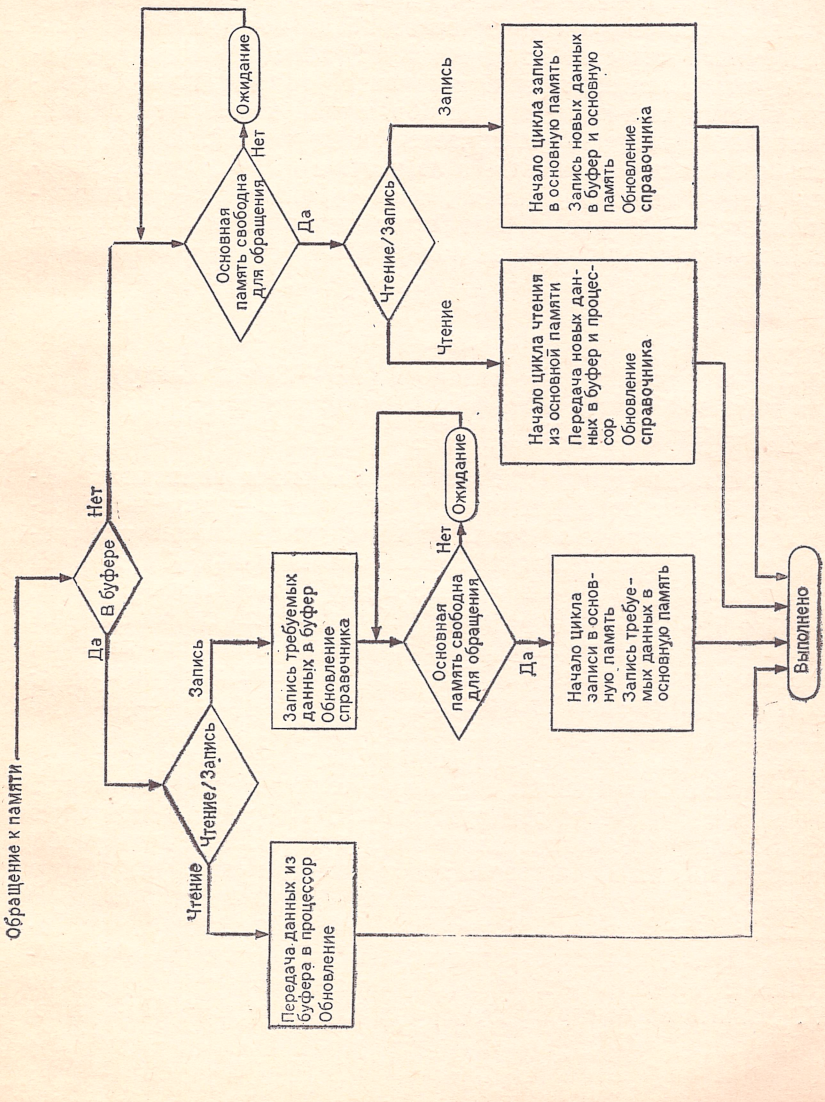


Рис. 3.4. Алгоритм сквозной записи (WT).

сложной реализацией данный алгоритм характеризуется концептуальной простотой, что обусловило его применение в различных системах с кэш-памятью.

Описанная схема может оказаться весьма ценной принадлежностью мультипроцессорных систем с раздельными кэшами и общей памятью. Путем применения алгоритма сквозной записи совместно с каким-либо механизмом фиксации содержимое основной памяти все время поддерживается в обновленном состоянии, и для отдельных процессоров отпадает необходимость в справочном поиске для нахождения последней копии или замены конкретного слова. Будучи концептуально простым, алгоритм сквозной записи имеет один существенный недостаток: в нем не заложена тенденция к минимизации доли обращений к основной памяти. При увеличении объема кэш-памяти доля обращений к основной памяти асимптоматически приближается к доле обращений для записи в основную память (а не к нулю) [B3]. Отсюда вытекает необходимость анализа более сложных схем организации записи. На рис. 3.4 представлено описание алгоритма сквозной записи в виде его блок-схемы.

3.3.3. Алгоритм простого свопинга (SS — Simple Swap) или обратной записи в конфликтных ситуациях (CUW — Conflicting Usage Writeback)

Данная схема имеет тенденцию к минимизации числа обращений к основной памяти за счет уменьшения числа запросов на доступ к основной памяти, приходящихся на каждое изменение содержимого кэш-памяти [B4]. Обращения к основной памяти имеют место в тех и только тех случаях, когда в быстром буфере не обнаруживается нужное слово. Эта схема свопинга повышает производительность системы памяти, так как в ней обращения к основной памяти не происходят при каждом запросе на запись, что имеет место при использовании алгоритма сквозной записи. Однако в связи с тем, что содержимое основной памяти не поддерживается в постоянно обновленном состоянии, если необходимого слова в быстром буфере не обнаруживается, из буфера в основную память надо возвратить какое-либо устаревшее слово, чтобы освободить место для нового необходимого слова. Поэтому из буфера в основную память сначала пересылается какое-то слово, место которого занимает в буфере нужное слово. Таким образом, происходят две пересылки между быстрым буфером и основной памятью. При использовании алгоритма WT данные из быстрого буфера в основную память не пере-

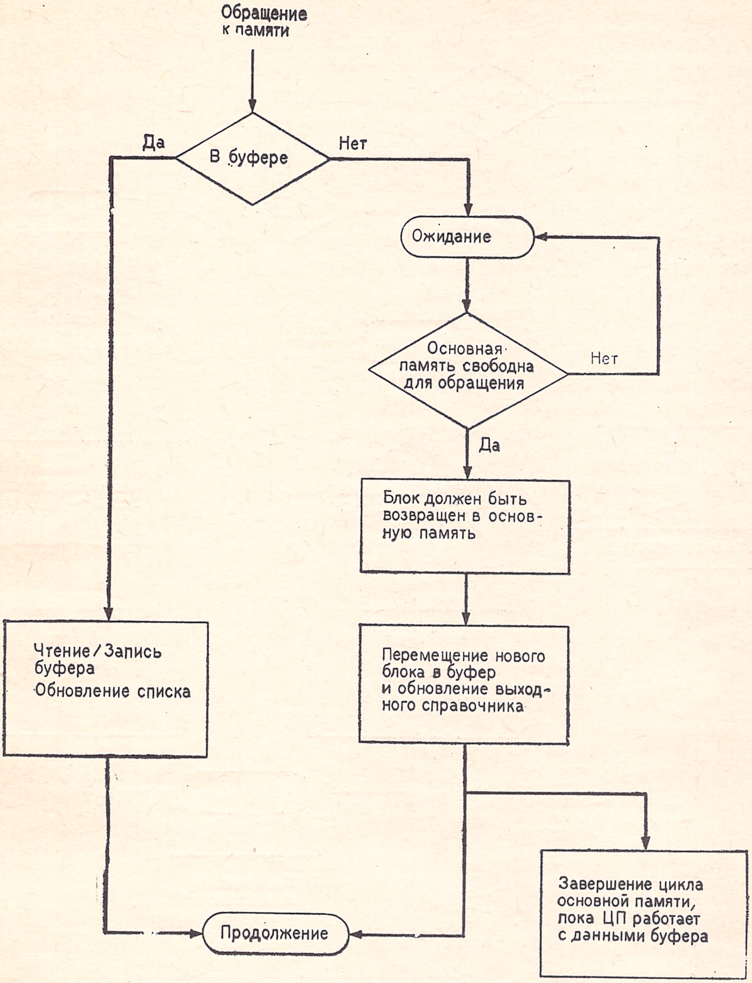


Рис. 3.5. Алгоритм простого свопинга с совмещением.

даются. Однако существенное преимущество алгоритма SS перед алгоритмом WT состоит в том, что доля обращений к основной памяти в общем числе обращений асимптотически приближается к доле обращений к памяти, при которых изменяется состав кэша, а не к доле обращений, в процессе которых осуществляется запись в память. Кроме того, если имеет место совмещение действий, ЦП может начинать выполнение операций сразу же после получения нового слова. Блок-схема алгоритма SS представлена на рис. 3.5.

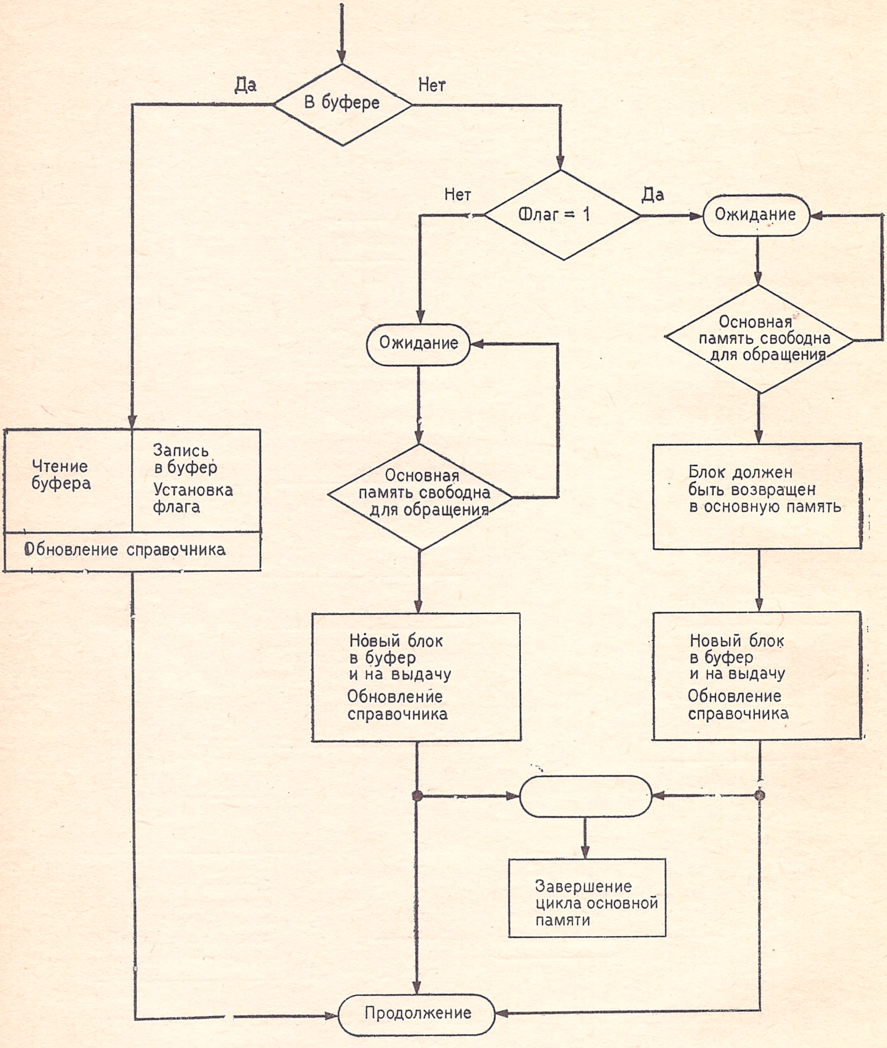


Рис. 3.6. Свопинг с флагами и совмещением.

3.3.4. Алгоритм свопинга с флагами (FS — Flagged Swap) или обратной записи в конфликтных ситуациях с флагами (CUX — Flagged Conflicting Writeback)

В алгоритме SS заложена тенденция к уменьшению эффективного времени цикла по сравнению с достигаемым при использовании алгоритма WT за счет уменьшения числа об-

ращений к основной памяти. Этот показатель можно снизить еще больше, если заложить в алгоритм выработку и учет определенной информации о состоянии слов, например, действительно ли необходимо перемещать из быстрого буфера в основную память слово, которое в дальнейшем должно подвергнуться этой пересылке. Если слово с того момента, как оно попало в буфер из основной памяти, не подвергалось изменениям, т. е. по его адресу не производилась запись (оно использовалось только для чтения), то нет необходимости пересылать его обратно в основную память, потому что в ней и так имеется достоверная его копия; это обстоятельство позволяет в ряде случаев обойтись без обращений к основной памяти. Если, однако, слово подвергалось изменениям с тех пор, когда его копия была в последний раз записана обратно в основную память, то приходится перемещать его в основную память. Отслеживать изменения слова достаточно просто, если пометить слово дополнительным флаговым битом. Изменяя значение флагового бита при изменении слова, можно сформировать информацию о состоянии слова; пересылать в основную память необходимо лишь те слова, флаги которых оказываются в установленном состоянии. Представляется, что алгоритм FS обладает достоинствами алгоритмов WT и SS. На рис. 3.6 показана последовательность действий при реализации алгоритма FS.

3.3.5. Алгоритм регистрового свопинга с флагами (FRS — Flagged Register Swap)

Еще один новый метод повышения производительности памяти за счет уменьшения эффективного времени цикла по сравнению с алгоритмом FS состоит во введении регистра (регистров) временного хранения между быстрым буфером и основной памятью. Теперь, если данные должны быть переданы из быстрого буфера в основную память, они сначала пересылаются в регистр (регистры) временного хранения; новое слово сразу же пересылается в буфер из основной памяти, а уже потом слово, временно хранившееся в регистре, записывается в основную память. Действия в ЦП начинают опять выполняться, как только для этого возникает возможность. Алгоритм FRS обладает всеми достоинствами алгоритма FS; кроме того, он обеспечивает совмещение операций записи в основную память с обычными операциями над буфером, что обеспечивает еще большее повышение производительности. На рис. 3.7 представлена блок-схема алгоритма FRS.

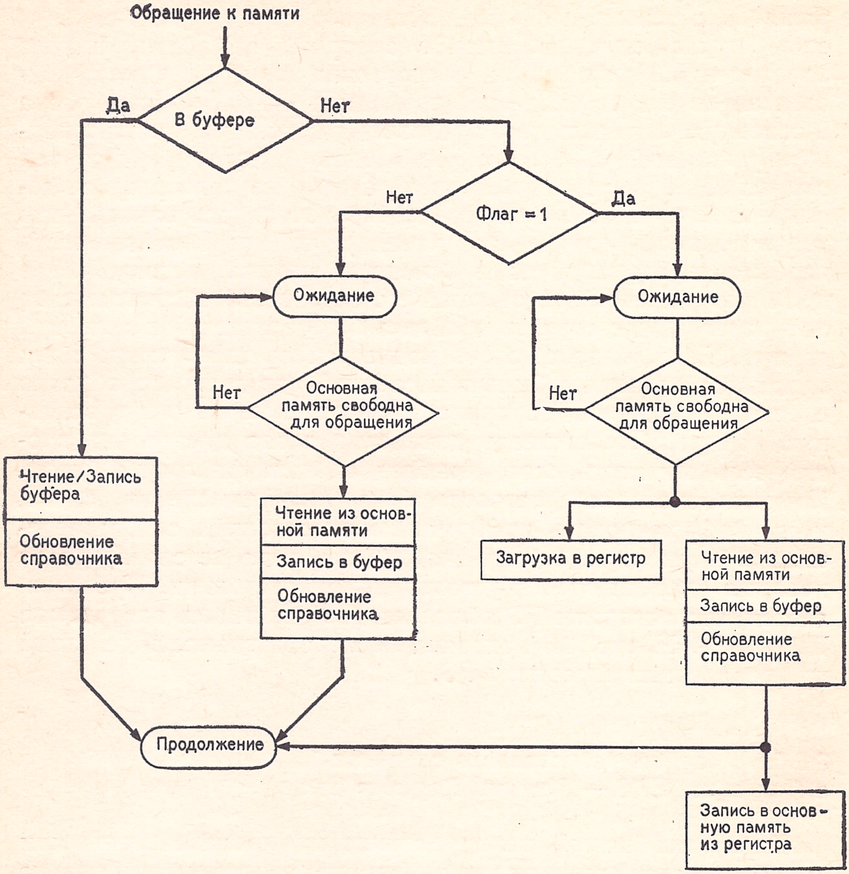


Рис. 3.7. Регистровый свопинг с флагами и совмещением.

3.4. СРАВНЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ СИСТЕМ ПАМЯТИ

Теперь оценим производительность систем памяти с буфером при использовании различных алгоритмов в предположении, что запросы на запись, неудачные обращения и слова, помеченные флагами, распределяются случайным образом, а операции над содержимым буфера совмещены с операциями основной памяти. Будут использованы следующие определенные ранее термины:

FR — доля операций чтения (в общем числе операций обращений к памяти),

HR — вероятность удачного обращения,

MR — вероятность неудачного обращения (1 — HR),

TSRC — время цикла чтения из основной (медленной) памяти,
TSRA — время обращения к основной (медленной) памяти при чтении,
TFRA — время обращения к буферу при чтении,
TDFR — время пересылки данных из основной памяти в буфер,
TFRC — время цикла чтения буфера,
TSRH — время поиска в справочнике,
DF — задержка на ожидание завершения предыдущей операции в основной памяти (состоящей из одного модуля),
X — доля блоков, отмеченных флагом,
TSWC — время цикла записи в основную (медленную) память,
TFWC — время цикла записи в буфер,
TSC — время цикла основной памяти, одинаковое при чтении и записи,
TFC — время цикла буфера, одинаковое при чтении и записи.

3.4.1. Временные соотношения для случая, когда память состоит из одного модуля

Производительность конкретной системы памяти с учетом задержки на ожидание высвобождения модуля основной памяти DF (предполагается, что она содержит один модуль) при использовании различных алгоритмов обращения рассчитывается следующим образом.

В системе памяти с буфером можно выделить четыре вида запросов:

запрос на чтение при нахождении требуемого слова (блоков) в буфере — удачное чтение (Rh);

запрос на чтение при отсутствии требуемого слова в буфере — неудачное чтение (Rm);

запрос на запись при нахождении требуемого слова в буфере — удачная запись (Wh);

запрос на запись при отсутствии требуемого слова в буфере — неудачная запись (Wm).

Эффективное время цикла можно определить, предположив, что все эти варианты запросов имеют место в соответствующей пропорции.

В приводимых ниже выражениях для обозначения операций умножения и возведения в степень использованы символы, применяемые как для традиционной, так и для машинной записи.

3.4.2. Временные соотношения для алгоритма сквозной записи

Twt = FR * HR * TFRC + (1 — HR) * FR * (TSRH + TSRA + DFA + TDFR) + (1 — FR) * TFWC или (1 — FR) * (DFA + Время запуска медленной памяти в зависимости от того, какое из этих времен больше).

[DFA — задержка ожидания единственного модуля при использовании алгоритма WT. Можно себе представить эту величину как состоящую из двух компонент: DFA чт. (задержка из-за чтения) и DFA зап. (задержка из-за записи).]

DFчт. = (1 — HR) * FR * (TSRC — TSRA — TDFR) + (1 — HR) * FR * (TSRC — TSRA — TDFR — TFRC) * (FR * HR) + (1 — HR) * FR * (TSRC — TSRA — TDFR — 2 * TFRC) * (FR * HR) * * 2 + ... до истечения времени задержки.

DFзап. = (1 — FR) * (TSWC — TFWC) + (1 — FR) * (TSWC — TFWC — TFRC) * (FR * HR) + (1 — FR) * (TSWC — TFWC — 2 * TFRC) * (FR * HR) * * 2 + ... до истечения составляющих времени задержки.

DFA чт. рассчитывается в предположении, что в основной памяти ранее выполнялась операция чтения, а DFA зап. — в предположении, что ранее в ней производилась запись. Кроме того, предполагается, что каждый новый записываемый блок заносится в буфер. Это может оказаться и нерациональным, особенно если используется прямой доступ к памяти.

3.4.3. Временные соотношения для алгоритма простого свопинга

Tss = FR * HR * TFRC + (1 — HR) * FR * (TSRH + TSRA + TDFR + DFB + TSWC) + (1 — FR) * HR * TFWC + (1 — FR) * (1 — HR) * (TSRH + TSRA + TDFR + DFB + TSWC).

DFB представляет собой задержку в случае применения алгоритма SS. Отметим, что время на поиски подходящего слова, подлежащего замещению, предполагается перекрывающимся с временем обращения к основной памяти и пересылки данных и поэтому не прибавляется в виде отдельной составляющей.

DFB = (1 — HR) * (TSRC — TSRA — TDFR) + (1—HR) * (TSRC — TSRA — TDFR — TFRC) * HR + (1 — HR) * (TSRC— TSRA — TDFR — 2 * TFRC) * (HR) * * 2 + ... до истечения составляющих времени задержки; для простоты предполагается, что временные циклы быстрого чтения и записи имеют одинаковую величину.

3.4.4. Временные соотношения для алгоритмов свопинга с флагами и регистрового свопинга с флагами

Tfs = FR * HR * TFRC + (1 — HR) * FR * (TSRH + TSRA + TDFR + DFR + X * TSWC) + (1 — FR) * HR * TFWC + (1 — FR) * (1 — HR) * (TSRH + TSRA + TDFR + DFC + X * TSWC) + ... до истечения составляющих времени задержки.

DFC — это такая же задержка, как и в случае алгоритма SS. Отметим, что время на поиски подходящего слова, подлежащего замещению, перекрывается с временем обращения к основной памяти и пересылки данных и поэтому не прибавляется в виде отдельной составляющей. DFC рассчитывается так же, как DFB.

Tfrs = FR * HR TFRC + (1 — HR) * FR * (TSRH + TSRA + (1 — X) * DFD + X * DFE + TDFR) + (1 — FR) * HR * TFWC + (1 — FR) * (1 — HR) * (TSRH + TSRA + (1 — X) * D FD + X * DFE + 1DFR)

DFD — это задержка в случае одномодульного варианта при использовании алгоритма FRS для чтения, а DFE — задержка для записи. DFD можно разбить на две составляющие: DFD чт. и DFD зап., которые рассчитываются, как и DFA чт. и DFA зап. DFE тоже можно разбить на две составляющие: DFEчт. и DFEзап., которые рассчитываются следующим образом.

DFEчт. = (1 — HR) * FR * (TSRC — TSRA — TDFR + TSWC) + (1 — HR) * FR * (TSRC — TSRA — TDFR + TSWC — TFRC) * (HR * FR) + (1—HR) * FR * (TSRC — TSRA— TDFR + TSWC — 2 * TFRC) (HR * FR) * * 2 + ... до истечения составляющих времени задержки + (1 — HR) * FR * (TSRC — TSRA — TDFR + TSWC — TFWC) * [(1 — FR) * HR] + (1 — HR) * FR * (TSRC — TSRA—TDFR + TSWC — 2*TFRC)[(1—FR)*HR]* * 2.

$$DFE_{зап.} = (1 - HR) * (1 - FR) * (TSRC - TSRA - TDFR + TSWC) + (1 - HR) * (1 - FR) * (TSRC - TSRA - TDFR + TSWC - TFWC) * (HR * FR) + (1 - HR) * (1 - FR) * (TSRC - TSRA - TDFR + TSWC - 2 * TFWC) * (HR * FR) * * 2 + \ldots \text{ до истечения составляющих времени задержки} + (1 - HR) * (1 - FR) * (TSRC - TSRA - TDFR + TSWC - TFWC * [(1 - FR) * HR] + (1 - HR) * (1 - FR) * (TSRC - TSRA - TDFR + TSWC - 2 * TFWC) * [(1 - FR) * HR] * * 2 + \ldots \text{ до истечения составляющих времени задержки.}$$

Используя совместно эти две формулы и полагая, что TFRC = TFWC, получаем

$$DFE = (1 - HR) * (TSRC - TSRA - TDFR + TSWC) + (1 - HR) * (TSRC - TSRA - TDFR + TSWC - TFRC) * HR + (1 - HR) * (TSRC - TSRA - TDFR + TSWC - 2 * TFRC) * (HR) * * 2 + \ldots .$$

В приведенных выше аналитических зависимостях расчет задержки производится только в тех случаях, когда имеют место неудачные обращения к буферу; при этом принимаются во внимание вероятности наличия тех или иных предыдущих последовательностей запросов к памяти и вычисляются результирующие задержки, определяемые этими последовательностями. Отметим, что если в последовательности обращений к памяти предыдущее обращение к буферу было удачным, то задержка в получении управления основной памятью снижается. Если, например, на три обращения к памяти раньше было необходимо обращаться к основной памяти (так как обращение к буферу оказалось неудачным), а два последующих обращения к памяти свелись к обращениям к буферу (эти обращения были удачными), то время, необходимое основной памяти для завершения выполнения операции, уменьшалось на величину двух циклов обращения к буферу.

На рис. 3.8 иллюстрируется ситуация, при которой последнее неудачное обращение к буферу имело место три запроса назад. Средние значения рассчитаны в предположении, что вероятность неудачного обращения представляет собой постоянную величину и неудачные обращения статистически независимы.

При рассмотрении всех этих систем памяти считается, что выполняется одно задание.

Рис. 3.9 иллюстрирует влияние применения разных алгоритмов записи на эффективное время цикла систем памяти при различных вероятностях удачного обращения и фиксиро-

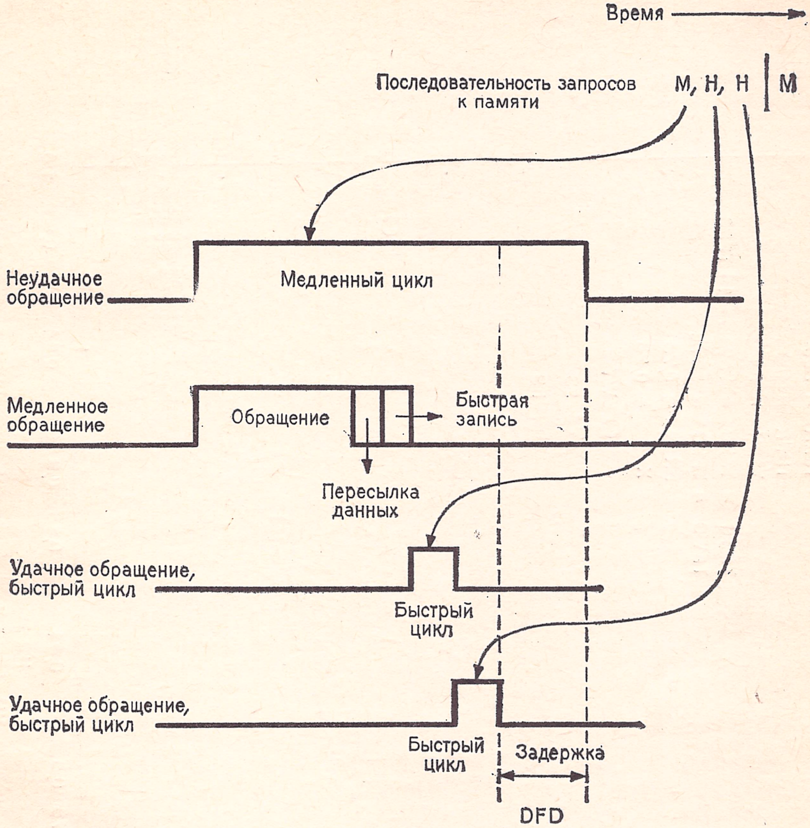


Рис. 3.8. Определение времени задержки.

ванных соотношениях операций чтения и записи. Эффективное время цикла определяется как среднее время, затрачиваемое на чтение или запись в память, в предположении, что всегда есть запросы на обращение к ней. Это предположение может не соответствовать действительности, так как время выполнения некоторых команд может превышать время цикла основной памяти. Результаты, представленные на рис. 3.9, являются наглядным доказательством преимуществ применения алгоритма FRS в моделируемой системе, состоящей из одного модуля основной памяти или множества модулей, работающих одинаковым образом.

На рис. 3.10 и 3.11 показана зависимость эффективного времени цикла от вероятности удачного обращения и доли операций чтения ($X = 0,25$) для времени цикла буфера 50 нс при цикле основной памяти 750 нс и времени цикла буфера

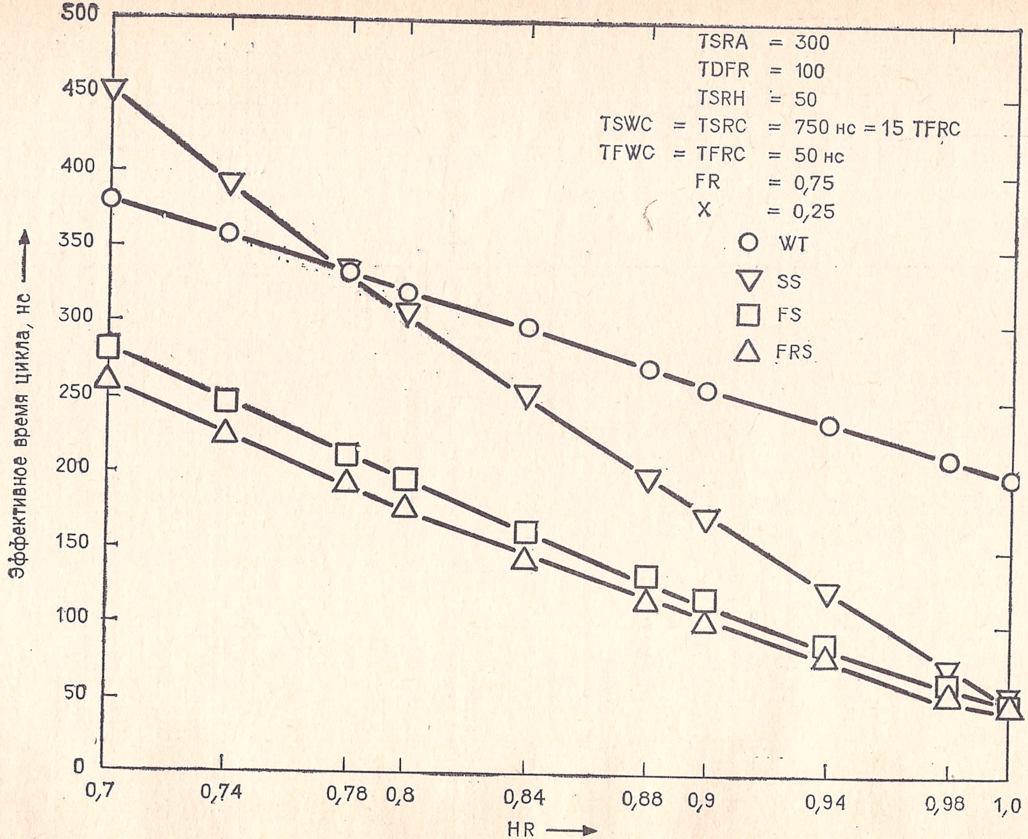


Рис. 3.9. Зависимость производительности системы памяти с буфером, имеющим время цикла 50 нс, от используемых алгоритмов перемещения данных.

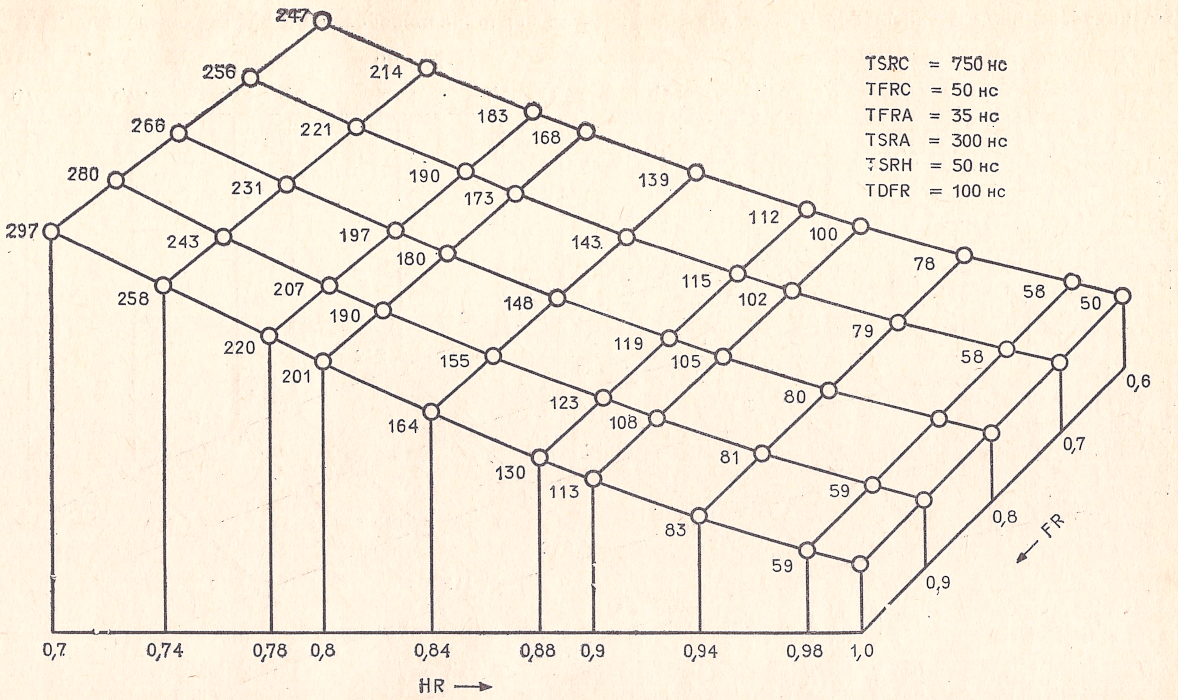


Рис. 3.10. Зависимость эффективного времени цикла от вероятности удачного обращения и доли операций чтения при времени цикла буфера 50 нс.

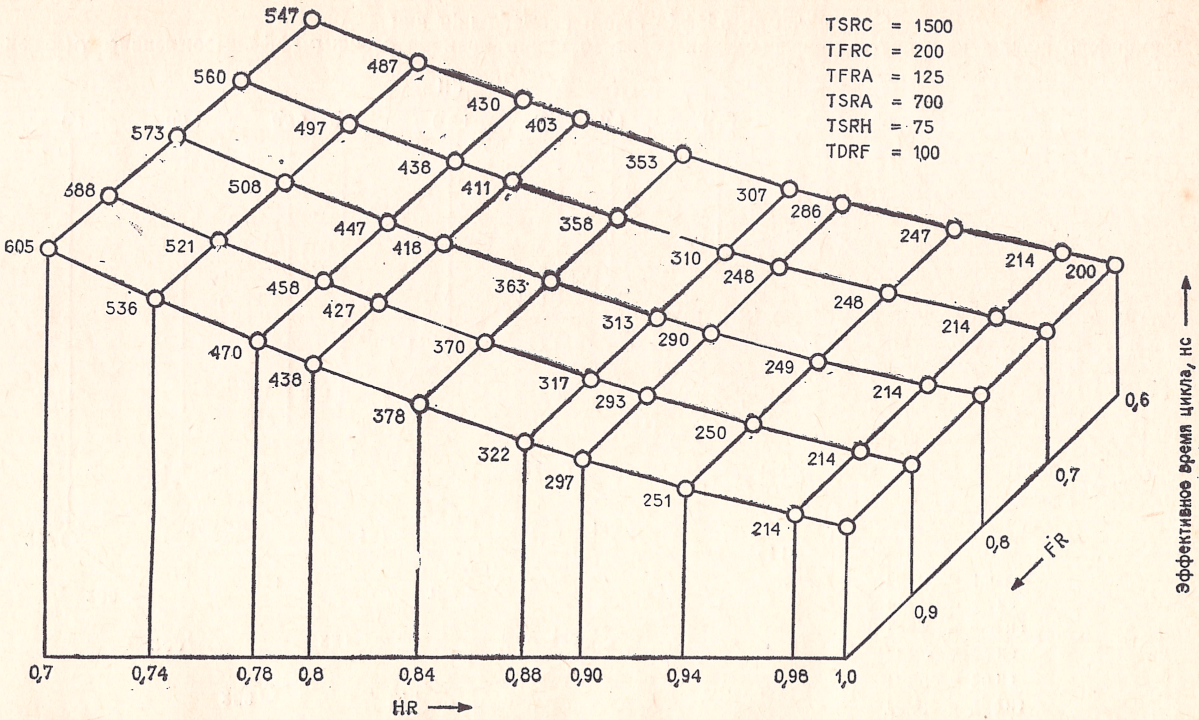


Рис. 3.11. Зависимость эффективного времени цикла от вероятности удачного обращения и доли операций чтения при времени цикла буфера 200 нс.

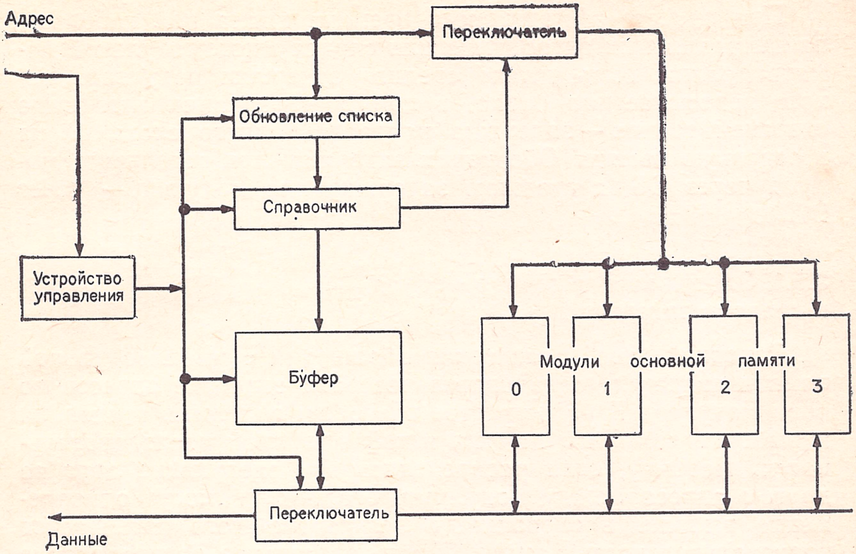


Рис. 3.12. Буферная система с многомодульной основной памятью.

200 нс при цикле основной памяти 1500 нс соответственно. Видно, что при использовании алгоритма FRS системы памяти нечувствительны к доле операций чтения. Кроме того, можно заметить, что быстродействие системы памяти превышает быстродействие основной памяти в 1—4 раза при вероятности удачного обращения, уменьшающейся до 0,80—0,82. Даже при столь низкой вероятности удачного обращения, как 0,7, может быть достигнуто уменьшение эффективного времени цикла по сравнению с циклом основной памяти в 2,5—3 раза.

3.5. СИСТЕМЫ С МНОГОМОДУЛЬНОЙ ОСНОВНОЙ ПАМЯТЬЮ

Если основная память состоит из нескольких модулей, которые могут функционировать независимо, то производительность алгоритма сквозной записи при использовании единственного буфера может быть улучшена потому, что несколько модулей одновременно могут быть заняты выполнением операций записи. Кроме того, такая организация памяти обеспечивает передачу информации от модулей основной памяти на шину в режиме разделения времени. На рис. 3.12 представлена структура системы памяти с буфером, содержащей многомодульную основную память.

При расчете эффективного времени цикла при использовании различных алгоритмов для случая многомодульной памяти должна быть учтена средняя задержка получения нового пересылаемого блока, а при использовании алгоритма сквозной записи — задержка на запись одного слова блока в основную память. Обозначим задержку для отдельного модуля через DS, а среднюю задержку для модулей — через DA.

3.5.1. Задержка для одного модуля и средняя задержка

Основной идеей расслоения обращений в многомодульной памяти является последовательное распределение адресов по всем модулям. В большинстве программ заложена тенденция к выборке команд и данных в той или иной последовательности. Поэтому, если распределить эти следующие друг за другом адреса между модулями, все модули могут одновременно выполнять операции, в результате чего повышается производительность памяти. Последовательное распределение адресов создает, кроме того, предпосылки для уменьшения числа конфликтов при обращении к памяти. На практике в большинстве программ выборка команд и данных не носит ни полностью последовательного, ни полностью беспорядочного характера. Вместо последовательного обращения к адресам, распределенным по разным модулям, время от времени может возникнуть несколько запросов подряд к одному и тому же модулю.

Состояния систем памяти с буфером при появлении запросов на чтение и запись могут быть более подробно классифицированы следующим образом:

запрос на чтение при нахождении запрашиваемого слова (блока) в буфере,
запрос на чтение при отсутствии слова в буфере,
запрос на запись при нахождении запрашиваемого слова в буфере,
запрос на запись при отсутствии слова в буфере.

Обычно все запросы, поступающие от центрального процессора или других процессоров, обрабатываются устройством управления памяти. Выполнив поиск в справочнике, устройство управления выясняет, к какому модулю памяти относится данный запрос, устанавливает тип запроса (на чтение или запись), а также определяет, находится ли запрашиваемое слово в буфере. Таким образом, конкретный запрос к памяти P^j может принадлежать к одной из следующих четырех категорий: P^j*чбп*, P^j*чбо*, P^j*збп*, P^j*збо*. Здесь j указывает номер модуля (от 1 до n), а индексы *чбп*, *чбо*, *збп* и *збо*

указывают тип запроса к памяти: чтение, блок присутствует; чтение, блок отсутствует; запись, блок присутствует; запись, блок отсутствует.

Таким образом, задержка DS применительно ко всем алгоритмам свопинга может быть по существу разделена на две части, одна из которых связана с предыдущими запросами на чтение (DSR), а другая — с предыдущими запросами на запись (DSW).

3.5.2. Расчет задержки при обращении к отдельному модулю (DS) с использованием алгоритма сквозной записи

Если обращение к памяти представляет собой запрос на чтение, а необходимые данные не находятся в буфере, для получения этих данных следует обратиться к одному из модулей основной памяти.

DSR представляет собой задержку в получении в распоряжение конкретного модуля памяти, вызванную предыдущим запросом на чтение. Так как причиной задержки является последовательность следующих друг за другом обращений к одному и тому же модулю памяти, можно рассчитать величину задержки, анализируя возможную последовательность предыдущих обращений. Число обращений к основной памяти зависит от используемого алгоритма управления памятью, поэтому DS тоже зависит от типа используемого алгоритма управления.

Предположим, что блок пересылается в буфер путем последовательной передачи слов из каждого модуля основной памяти. Система может либо ожидать окончания пересылки всех слов блока, либо продолжить работу, как только будет получено желаемое слово. Хотя во втором случае и достигается некоторое повышение производительности, соответственно усложняется и полный алгоритм управления. Ниже приводится пример расчета для более сложного случая с тем чтобы продемонстрировать набор процедур, используемых при вычислении задержек. Величина $((N+1)/2*TDFR)$ представляет собой среднее время пересылки данных по однословной шине. В идеальном случае было бы желательно иметь возможность подать следующий запрос сразу же по истечении этого времени. Однако другой запрос не может быть обслужен, пока не закончится цикл памяти. Так как для завершения пересылки блока, очевидно, требуется дополнительное время, величина задержки для одного модуля, являющейся следствием предыдущего запроса на чтение блока

из него, составляет

$$TSRC - TSRA - ((N + 1)/2 * TDFR)).$$

Множитель $(N + 1)/2$ позволяет учесть то обстоятельство, что нужный модуль в процессе пересылки может оказаться первым, вторым и т. д.

Средняя задержка на один модуль зависит от доли операций записи, вероятности неудачных обращений, времени доступа и времени пересылки; все эти факторы должны быть учтены при расчете величины задержки.

Если последовательность запросов к памяти включает запросы на чтение блока, а затем запросы на запись блока, то вероятность появления задержки определяется формулой

$$FR(1 - HR) * FR\,HR.$$

Последовательность запросов может состоять из запроса на чтение к какому-то модулю памяти и еще одного запроса на чтение к тому же модулю, при котором произошло неудачное обращение к буферу.

Задержка, т. е. время ожидания, являющееся следствием такого сочетания следующих друг за другом запросов, равна

$$TSRC - TSRA - (N + 1)/2 * TDFR - TFRC.$$

Образование задержки поясняется графически на рис. 3.13. Обратите внимание, что TFRC (цикл чтения быстрого буфера) вычитается из времени задержки, потому что цикл чтения буфера может быть начат до окончания предыдущего цикла основной памяти. Время задержки для одного модуля уменьшается на эту величину.

Таким образом, интервал $(TSRC - TSRA - (N + 1)/2 * * TDFR)$ может наложиться на время выполнения запросов на чтение информации, находившейся в буфере. Если последовательность запросов к памяти имеет вид

$$\textit{чбо}, \underbrace{\textit{чбп}, \ldots, \textit{чбп}}_{K}$$

и при этом $(TSRC - TSRA - (N + 1)/2 * TDFR) \leqslant K * TFRC$, то все K запросов на чтение информации, находящейся в буфере, могут быть совмещены во времени.

3.5.3. Расчет DSR при использовании алгоритмов WT, SS, FS и FRS

Величина DSR зависит от типа алгоритма управления, так как тип алгоритма влияет на число обращений к основной памяти.

При выполнении операции *чбо* в случае использования алгоритма WT необходимо всего одно обращение к основной памяти (для взятия из нее требуемых данных), потому что содержимое основной памяти постоянно поддерживается в обновленном состоянии и нет необходимости перемещать в нее какую бы то ни было информацию из буфера. При использовании алгоритма SS обращаться к основной памяти приходится дважды: один раз для пересылки (чтения) блока данных в буфер и один раз для пересылки (записи) необходимого блока данных из буфера в основную память. В случае использования алгоритмов FS и FRS число обращений к основной памяти при выполнении операции *чбо* зависит от значения флагового бита. Если флаговый бит блока слов, подлежащего замещению, установлен, значит слова этого блока подвергались изменениям после того, как он был перемещен в буфер из основной памяти. Следовательно, копия этого блока, находящаяся в основной памяти, уже не является достоверной, и блок надо переслать в основную память до того, как на его место в буфер будет переслан новый необходимый блок. Таким образом, оказываются необходимыми два обращения к основной памяти. Если же флаговый бит блока,

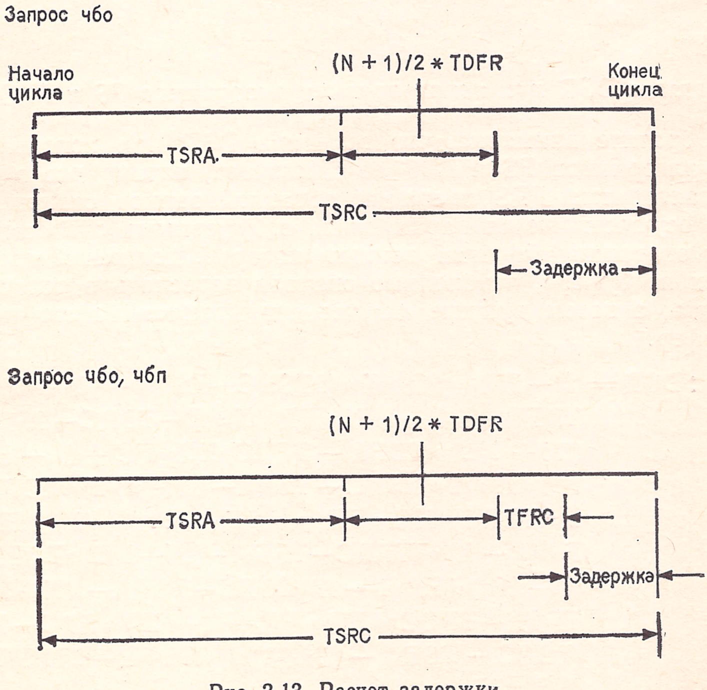


Рис. 3.13. Расчет задержки.

подлежащего замещению, сброшен, значит, блок с момента пересылки его в буфер изменениям не подвергался. Задержка, связанная с доступом к одному модулю в различных ситуациях, может быть рассчитана следующим образом:

WT : DSR = FR (1 — HR) (TSC — TSRA — ((N+1)/2) * TDFR) + FR(1 — HR)(TSC — TSRA — ((N + 1)/2) * TDFR— TFRC) * FR * HR + FR (1 — HR) (TSC — TSRA — ((N + 1)/2) * TDFR — 2TFRC) (FR * HR) **2 + ...

Рассуждая так же, как и в случае алгоритма WT, можно получить выражения для DSR применительно к алгоритмам SS, FS и FRS.

SS : DSR = (1 — HR) * FR * (2 * TSC — TSRA — ((N + 1)/2) * TDFR) + (1 — HR) FR * (2 * TSC — TSRA — ((N + 1)/2) * TDFR — TFRC) * HR * FR +(1—HR) FR * (2 * TSC—TSRA—((N+1)/2)*TDFR—2 * TFRC) * (HR * FR) * * 2 + ... до истечения составляющих времени задержки.

Задержка DSR *при использовании алгоритмов* FS *и* FRS

Обратите внимание, что величина DSR при использовании алгоритмов FS и FRS не одинакова, потому что в случае алгоритма FRS операции обратной записи совмещены, т. е. осуществляются параллельно с буферными операциями. При операции *чбо* блок данных, который должен быть выведен из буфера, необходимо сначала переслать в основную память, а затем нужные данные считываются на его место в буфер.

FRS : DSR = (1 — HR) * FR * X * (2TSRC—TSRA—((N + 1)/2) * TDFR) + (1 — HR) * FR * X * (2—TSRC—TSRA— ((N + 1)/2) * TDFR — TFRC) HR * FR + (1 — HR)* FR * X * (2 * TSRC — TSRA— ((N+1)/2) * TDFR — 2 * TFRC) (HR * FR) * * 2 + ... + (1 — HR) * FR (1 — X) * (TSRC—TSRA—((N+1)/2) *TDFR)+ (1 — HR) * FR (1 — X) (TSRC—TSRA— ((N+1)/2 * TDFR—TFRC) * HR * FR+(1—HR) * FR(1—X) (TSRC — TSRA — ((N+1)/2) * TDFR — 2TFRC) (HR * FR) * * 2 + ...

Задержка DSR *при использовании алгоритма* FS

При использовании алгоритма свопинга с флагом запись в случае необходимости завершается до того, как будет выполняться чтение. Величина задержки определяется выражением

$$FS : DSR = (1 - HR) * FR * \Big(TSRC - TSRA - \frac{(N+1)}{2} TDFR\Big) + (1 - HR) * FR * \Big(TSRC - TSRA - \frac{(N+1)}{2} TDFR - TFRC\Big) * HR.$$

3.5.4. Максимальная средняя задержка модуля

Максимальная средняя задержка для всех модулей (DA) может быть рассчитана по существу тем же способом, что и задержка для одного модуля. DA тоже можно разделить на DAR и DAW. Задержка DA зависит от алгоритмов свопинга и алгоритмов записи. Она может быть определена следующим образом:

$$WT : DAR = (1 - HR) * FR * (TSC - TSRA - TDFR) + (1 - HR) * FR * (TSC - TSRA - TDFR - TFRC) * (HR * FR) + (1 - HR) * FR * (TSC - TSRA - TDFR - 2.TFRC) * (HR + FR) * * 2 + \ldots .$$

$$SS : DAR = (1 - HR) * FR (TSC - TSRA - TDFR) + (1 - HR) * FR (TSC - TSRA - TDFR - TFRC) * (HR * FR) + (1 - HR) * FR (TSC - TSRA - TDFR - 2 * TFRC) * (HR * FR) * * 2 + \ldots .$$

$$FS : DAR = FR * (1 - HR) * X * (TSC - TSRA - TDFR) + FS * (1 - HR) * X * (TSC - TSRA - TFRC) * (HR) + FR * (1 - HR) * X * (TSC - TSRA - TDFR - 2 * TFRC) * (HR) * * 2 + \ldots + FR * (1 - HR) * (1 - X) * (TSC - TSRA - TDFR) + FR * (1 - HR) * (1 - X) * (TSC - TSRA - TDFR - TFRC) * HR + FR * (1 - HR) * (1 - X) * (TSC - TSRA - TDFR - 2 * TFRC) * HR * * 2 + FR * (1 - HR) * (1 - X) * (TSC - TSRA - TDFR - 2 * TFRC) * (HR) * * 2 + \ldots .$$

$$FRS : DAR = FR * (1 - HR) * X * (2TSC - TSRA - TDFR) + FR * (1 - HR) * X * (2TSC - TSRA - TDFR - TFRC) * (HR) + FR * (1 - HR) * X * (2TSC - TSRA -$$

TDFR—2 * TFRC) * (HR) * * 2 + FR * (1—HR) * (1 — X) * (TSC — TSRA — TDFR)+FR * (1—HR) * (1 — X) * (TSC — TSRA — TDFR — TFRC) * HR+ FR * (1 — HR) * (1 — X) * (TSC — TSRA—TDFR— 2 * TFRC) * (HR) * * 2 + ...

Расчет DSW

Еще одним фактором, влияющим на среднюю задержку отдельного модуля в многомодульной системе, является задержка, связанная с неудачным предыдущим обращением для записи. DSW тоже зависит от используемого типа алгоритма управления.

Алгоритм WT

При использовании алгоритма WT любая операция записи в память включает запись в основную память независимо от того, находится нужное слово (блок) в буфере или нет. Поэтому применительно к этому алгоритму понятия задержки при *збп* и *збо* могут быть объединены и рассматриваться просто как задержка при записи. Так как время цикла записи в буфер значительно меньше времени цикла записи в основную память, система должна ожидать, пока не будет инициирована операция с основной памятью.

Рассмотрим последовательность запросов к памяти, в которой запрос на запись сопровождается запросом на чтение, сразу после которого имеет место интересующий нас запрос. Вероятность того, что нужный модуль занят, задается выражением

((1 — FR)/N) * (HR * FR).

Время ожидания этого конкретного модуля составляет

(TSWC — TSWS — TFRC),

где TSWS — время загрузки при записи в основную память; оно может быть значительно меньше времени полного цикла записи.

Параметр TFRC входит в выражение со знаком минус, так как цикл чтения буфера может быть начат сразу по окончании TSWS.

Поэтому если последовательность запросов памяти имеет вид

$$\textit{зап}, \underbrace{\textit{чбп}, \ldots, \textit{чбп}}_{K}$$

и, кроме того, если $(TSWS - TSWC) > K \cdot TFRC$, то K предыдущих запросов на чтение блока могут быть совмещены по времени с предыдущей операцией обращения к памяти и эффективная задержка для отдельного модуля снижается. Таким образом, при использовании алгоритма сквозной записи

$$DSW = ((1-FR)/N) * (TSWC-TSWS) + ((1-FR)/N) * (TSWC - TSWS - TFRC) * HR * FR + ((1-FR)/N * (TSWC - TSWS - 2TFRC) * (HR * FR) * * 2 + \ldots + ((1-FR)/N) * ((1-FR)(N-1)/N) * (TSRC - TSWS - TFRC) + ((1-FR)/N) * ((1-FR)(N-1)/N)^2 * (TSRC - TSWS - 2TFRC).$$

Алгоритмы SS, FS *и* FRS

При использовании алгоритмов SS, FS и FRS обращения к основной памяти для записи происходят лишь в тех случаях, когда запрашиваемое слово (блок) не обнаруживается в быстром буфере. В случае алгоритма SS число обращений к основной памяти при операции *збо* равно двум, а при использовании алгоритмов FS и FRS оно зависит от вероятности того, что блок отмечен флагом. Приводимые ниже выражения характеризуют величины задержек, имеющих место при обращении к модулю, содержащему часть всего блока. В процессе реализации данных алгоритмов множество модулей функционирует одинаковым образом, и не имеет смысла продолжать операции, пока не завершатся пересылки всех модулей. Предполагается, что $TFWC = TFRC$.

$$SS: DSW = DAW = ((1-FR) * (1-HR)(TSRC - TFWC - NTDFR) + ((1-FR) * (1-HR) * (TSWC - TFWC - TFWC - NTDFR) HR + ((1-FR) * (1-HR) * (TSWC - TFRC - 2TFWC - NTDFR) HR * * 2 + \ldots .$$

$$FS: DSW = DAW = ((1-FR)(1-HR) * X * (TSRC - TFWC - NTDFR) + ((1-FR)(1-HR) * X * (TSRC - TFRC - TFWC - NTDFR) HR + ((1-FR) * (1-HR) * X * (TSRC - TFRC - 2TFWC - NTDFR) HR * * 2 + \ldots + ((1-FR) * (1-HR) * (1-X)(TSRC - TFWC - NTDFR) + (1-FR) * (1-HR) * (1-X)(TSRC - TFRC - TFWC - NTDFR) HR + ((1 - FR (1-HR) * (1-X)(TSRC - TFRC - 2TFWC - NTDFR) HR * * 2 + \ldots .$$

Аналогично может быть вычислена величина DSW для случая применения алгоритма FRS.

FRS : DSW = DAW = ((1 — FR) * (1—HR) * X (TSRC—TFWC—NTDFR) + ((1 — FR) * (1 — HR) * X (TSRC — TFRC — TFWC — NTDFR) HR + ((1 — FR) * (1 — HR) * X (TSRC — TFRC — 2TFWC — NTDFR) HR^2 + . . . + . . . + ((1 — FR)/N * (1 — HR) * (1 — X) * (TSRC—TFWC—NTDFR)+((1—FR)/N)* (1 — HR) * (1 — X) * (TSRC — TFRC — TFWC — NTDFR) HR + ((1 — FR)/N) * (1 — HR) * (1 — X) * (TSRC — TFRC — 2TFWC — NTDFR) HR * * 2.

3.6. ЭФФЕКТИВНЫЕ ВРЕМЕНА ЦИКЛА И ВЫБОРКИ ИНФОРМАЦИИ

Располагая величинами задержки одного модуля и средней задержки, можно рассчитать эффективные времена цикла и выборки для системы памяти. Эффективное время цикла — это среднее время чтения слова из кэш-памяти и записи слова в эту память; рассчитать его можно как величину, среднюю для следующих четырех ситуаций:

чтение при отсутствии блока в буфере,
чтение при нахождении блока в буфере,
запись при отсутствии блока в буфере,
запись при нахождении блока в буфере.

Эффективное время цикла для четырех алгоритмов свопинга определяется следующим образом:

WT cycle = FR * HR * TFRC+(1—HR) * FR * (TSRH+TSRA + DA + NTDFR) + (1 — FR) (DS + TSWS)

SS cycle = FR * HR * TFRC+(1 — HR) * FR * (TSRH+TSRA + NTDFR + TSWC + DA) + (1 — FR) * HR * TFWC+ (1 — FR) * (1 — HR) * (TSRH + TSRA + NTDFR + TSWC + DA) + (1 — FR * (1 — HR) * (TRSH + TSRA + NTDFR + TSWC + DA)

FS cycle = FR*HR*TFRC + (1 — HR)*FR*(TSRH + TSRA + NTDFR + XTSWC+DA)+(1—FR) * HR * TFWC + (1 — FR) * (1 — HR)(TSRH + XTSWC + TSRA + NTDFR + DA)

FRS cycle = FR * HR * TFRC+(1—HR) * FR * (TSRH+TSRA+ DA + NTDFR) + (1 — FR) HR * TFWC + (1 — FR) (1 — HR) * (TSRH + TSRA + DA + NTDFR)

На рис. 3.14 показана зависимость средней задержки обращения к отдельному модулю от числа модулей при использовании алгоритма сквозной записи. Из графиков видно, что задержку можно снизить за счет увеличения числа модулей. На рис. 3.15 показана зависимость этой задержки от вероятности удачного обращения и доли операций чтения. На рис. 3.16—3.18 графически представлены эти зависимости при отношении быстродействия быстрой памяти к быстродействию медленной памяти, равном соответственно 4, 8 и 16, при использовании алгоритма сквозной записи. Эти графики позволяют, кроме того, судить о чувствительности алгоритма сквозной записи к доле операций чтения.

На рис. 3.19—3.21 графически представлены зависимости эффективного времени цикла от вероятности удачного обращения и доли операций чтения при отношении быстродействия быстрой и медленной памяти, составляющем 4, 8 и 16 соответственно, при использовании алгоритма FRS. Обратите внимание, что эффективное время цикла для данного алгоритма почти не зависит от доли операций чтения.

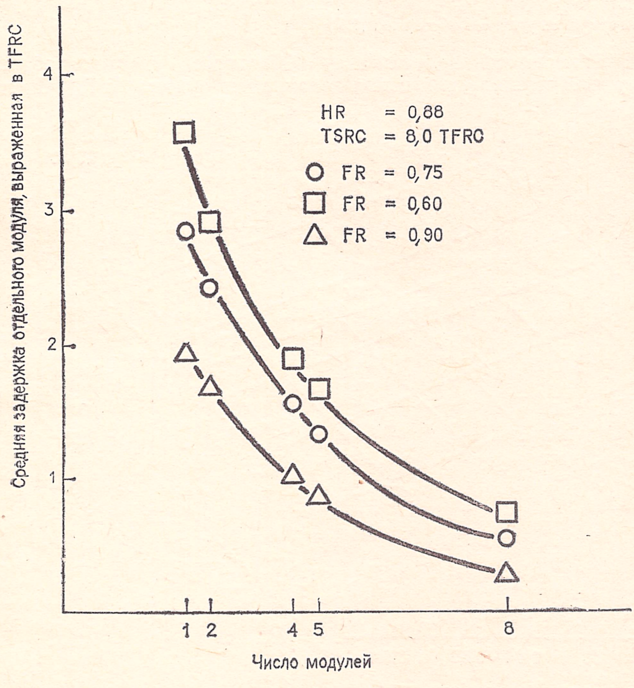


Рис. 3.14. Влияние числа модулей на задержку при использовании алгоритма сквозной записи.

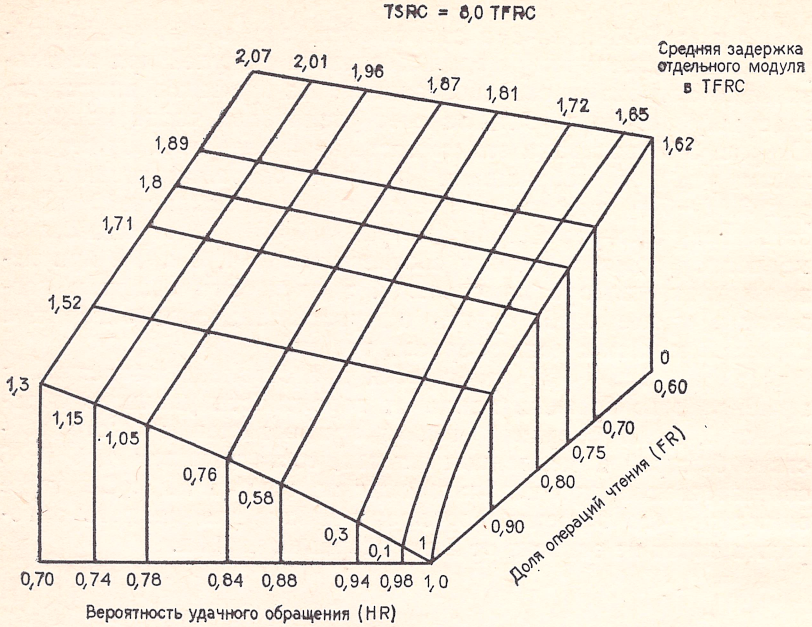


Рис. 3.15. Зависимость задержки от вероятности удачного обращения и доли операций чтения при использовании алгоритма WT. (Число модулей = 4.)

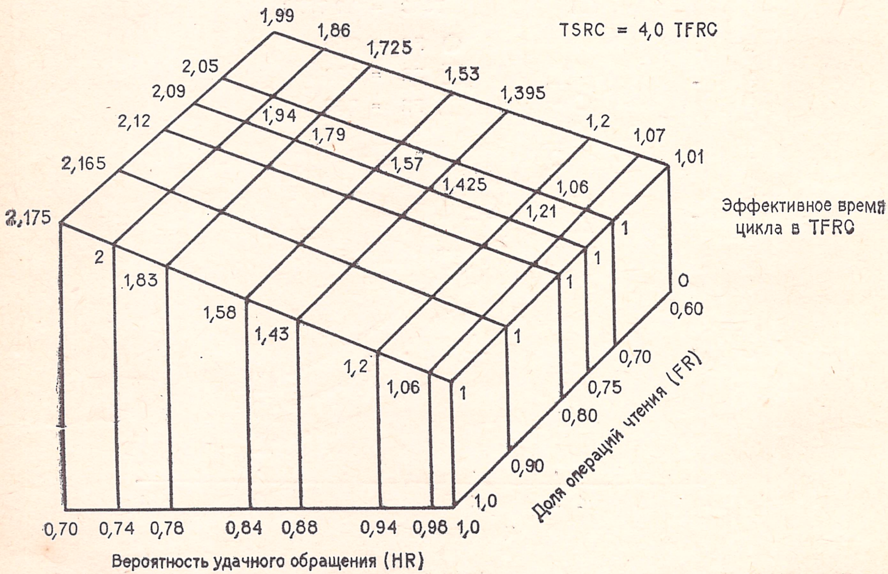


Рис. 3.16. Зависимость эффективного времени цикла от вероятности удачного обращения и доли операций чтения при использовании алгоритма WT. (Число модулей = 4.)

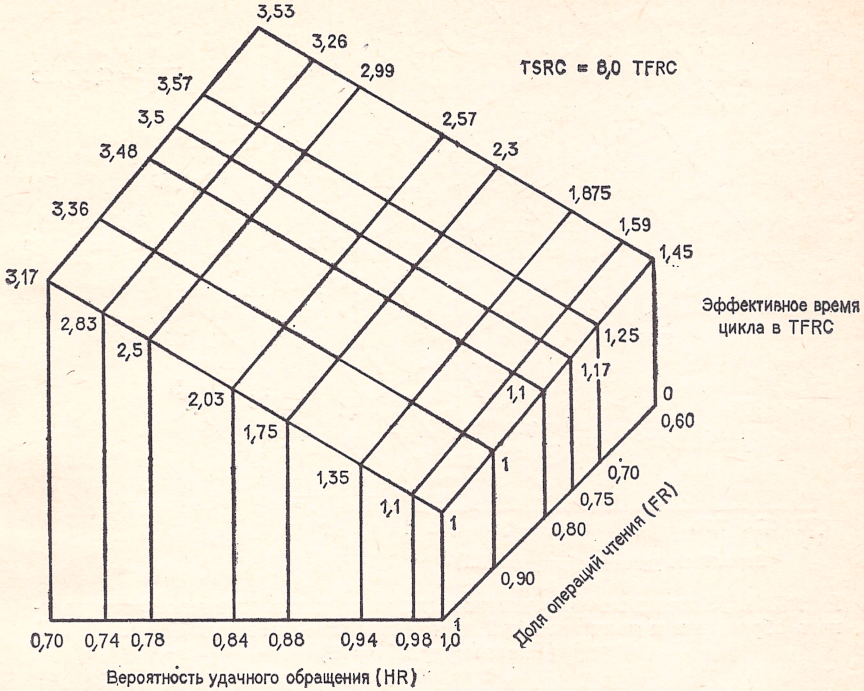


Рис. 3.17. Зависимость эффективного времени цикла от вероятности удачного обращения и доли операций чтения при использовании алгоритма WT. (Число модулей = 4.)

УПРАЖНЕНИЯ

1. Предположим, что для организации работы кэш-памяти используется алгоритм свопинга с флагами, причем HR = 0,95, X = 0,2, TFC = 0,1 мкс, TSRC = TSWC = 1,0 мкс, TSRA = = 0,6 мкс. Определите эффективное время цикла памяти с совмещением и без совмещения кэш-операций с завершающей частью цикла основной памяти.

2. Пусть для обслуживания кэш-памяти используется алгоритм сквозной записи, причем FR = 0,8, HR = 0,95, TFC = = 0,1 мкс, TSRC = TSWC = 1,0 мкс. Определите, какая часть запросов к основной памяти связана с операциями записи в память, а какая — с неудачными обращениями к буферу в ходе операций чтения.

3. Определите разницу в эффективном быстродействии систем памяти, содержащих кэши с параметрами, заданными в упр. 2, если в одной из этих систем при неудачной записи блок переносится в буфер, а в другой — нет.

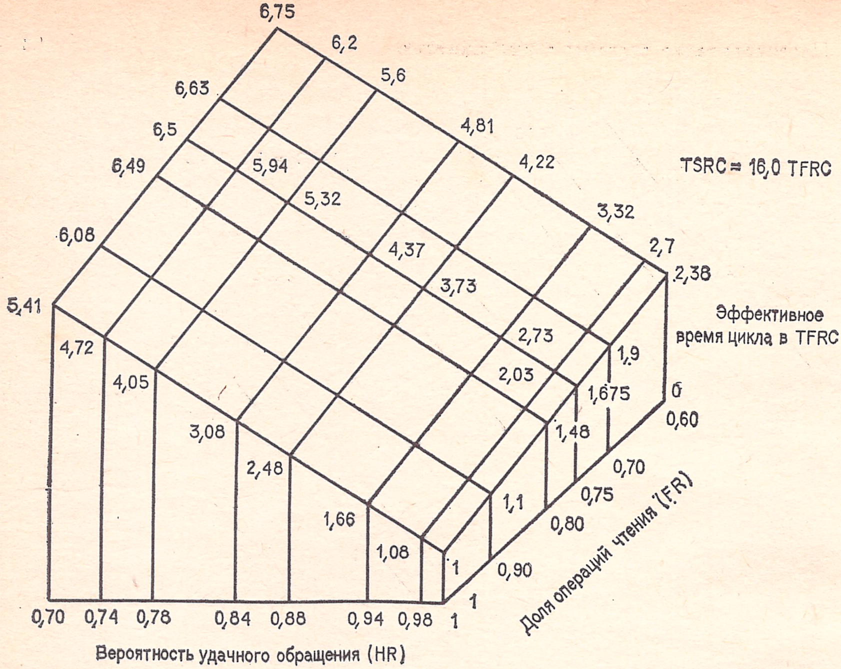


Рис. 3.18. Зависимость эффективного времени цикла от вероятности удачного обращения и доли операций чтения при использовании алгоритма WT. (Число модулей = 4.)

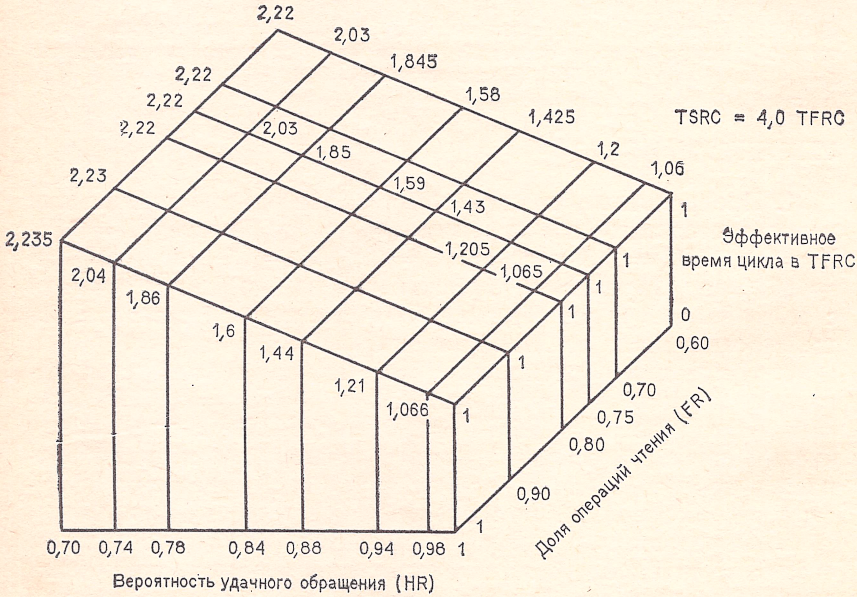


Рис. 3.19. Зависимость эффективного времени цикла от вероятности удачного обращения и доли операций чтения при использовании алгоритма FRS. (Число модулей = 4.)

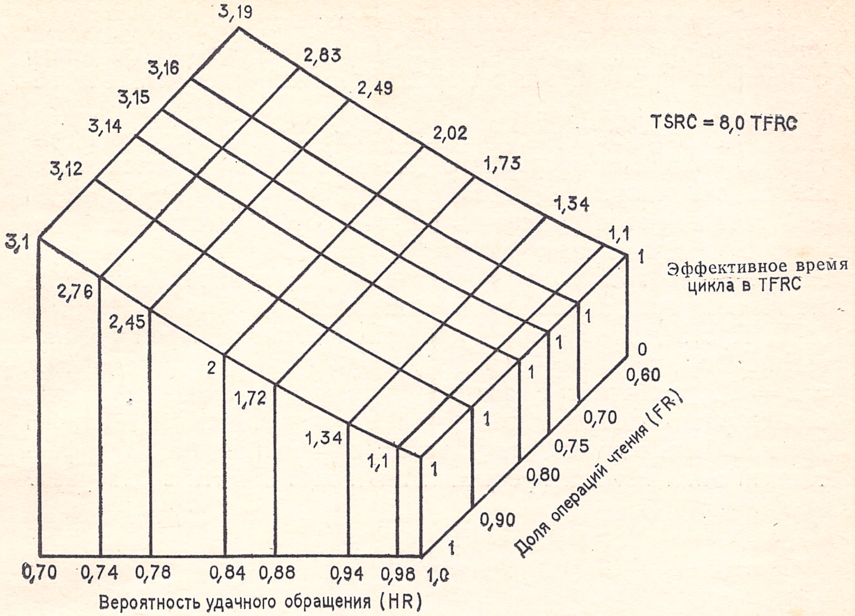


Рис. 3.20. Зависимость эффективного времени цикла от вероятности удачного обращения и доли операций чтения при использовании алгоритма FRS. (Число модулей = 4.)

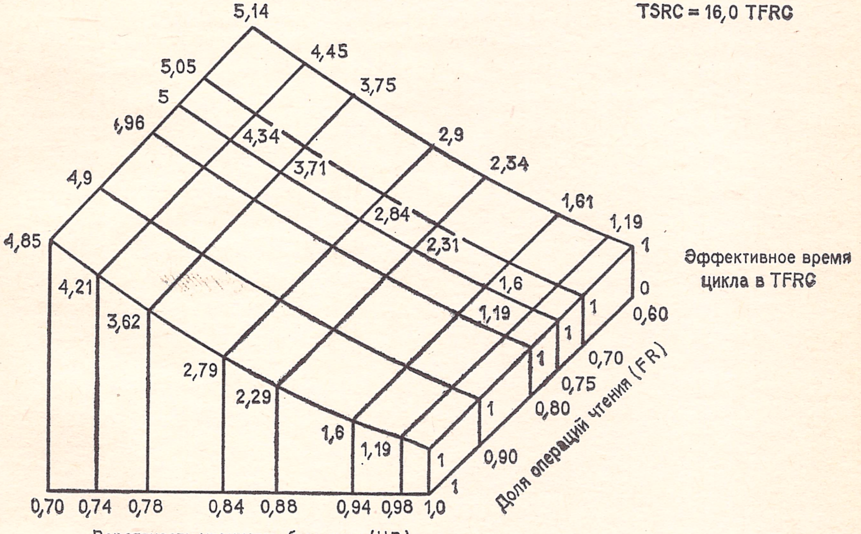


Рис. 3.21. Зависимость эффективного времени цикла от вероятности удачного обращения и доли операций чтения при использовании алгоритма FRS. (Число модулей = 4.)

4. Определите эффективное время цикла для условий упр. 1 в предположении, что вместо алгоритма свопинга с флагами применяется алгоритм простого свопинга без перекрытия.

5. При расчете эффективного времени цикла в упр. 1 считалось, что новая операция с кэш-памятью начинается сразу же по окончании текущего цикла. Предположим, что кэш работает при этом совместно с процессором, который инициирует запрос к памяти каждые 0,3 мкс. Определите среднюю задержку доступа к основной памяти, если имеет место совмещение операций обращения к памяти.

6. Пусть кэш-память, функционирование которой организовано по алгоритму свопинга с флагами без перекрытия, а $x = 0{,}2$, работает с основной памятью, имеющей цикл, в 100 раз более длинный, чем цикл кэш-памяти. Какова должна быть вероятность удачного обращения, чтобы эффективный цикл памяти превышал цикл кэш-памяти в два раза?

Глава 4

Вероятность удачного обращения влияющие на нее факторы

4.1. АНАЛИТИЧЕСКИЕ МОДЕЛИ

Рассматриваемая здесь аналитическая модель позволяет понять суть вероятности удачного обращения к памяти с буферизацией, она используется в качестве простого средства анализа систем памяти. Для упрощения и удобства дискретные распределения вероятностей аппроксимируются, где это возможно, непрерывными функциями. Более формальное изложение соответствующих вопросов можно найти в работе [E2].

Так как новые сегменты задания подвергаются диспетчеризации после выполнения, очевидно, что набор тех слов из числа находящихся в основной памяти, которые используются наиболее часто в пределах некоторого временного интервала, меняется во времени. Отсюда вытекает упрощающее предположение о том, что частота использования слов и блоков является функцией, стационарной во времени. С аналитической точки зрения удобно считать, что если буфер полон, то частота использования блока является функцией, стационарной во времени; зависимость от времени можно ввести, предположив, что подмножество используемых блоков памяти выделяется в начале реализации задания и остается впоследствии фиксированным. Затем можно отразить эффект переключения заданий, рассматривая процесс заполнения буфера и ограничивая продолжительность выполнения задания. Переход к выполнению новых сегментов программы аналогичен переключению заданий.

Для целей анализа будем считать, что частота использования хранящихся в памяти блоков (имеющих по одному или по нескольку слов) и соответствующего содержимого буфера описывается модифицированной моделью независимых обращений (марковской цепи), как это было предложено Истоном [E2], т. е. определим P_{ij} как вероятность того, что после того, как был выбран i-й блок, следующим будет выбран j-й блок:

$$P_{ij}=(1-r)\lambda_j, \quad P_{ii}=r\lambda_i+(1-r)\lambda_i;$$

здесь λ_i или λ_j — вероятности соответственно того, что i-й и j-й блоки окажутся объектами очередного независимого обращения, а r — вероятность того, что сразу же будет иметь место обращение к i-му блоку в соответствии с моделью цепи

Маркова. Усредненная по времени вероятность Z_i выбора i-го блока задается просто как $Z_i = \lambda_i$, даже если имеет место обращение.

Однако вследствие того, что существует вероятность r того, что только что выбранный блок сразу же будет объектом очередного обращения до того, как произойдет независимое обращение, вероятность неудачного обращения к буферу снижается. Слово, к которому только что имело место обращение, находится в буфере, и при немедленном повторном обращении к этому слову (предполагается, что используется стратегия замещения LRU или FIFO[1]) не происходит ни неудачного обращения, ни изменения порядка размещения информации в стеке. Вероятность неудачного обращения M'_r для модифицированной модели независимых обращений в состоянии равновесия выражается через вероятность M_r для модели независимых обращений с помощью следующего простого соотношения:

$$M'_r = (1 - r) M_r,$$

так как доля неудачных обращений не может превышать $1 - r$.

Если величина M_r определена для случая независимых обращений ($r = 0$), то выражение для модифицированного случая получается просто путем включения коэффициента $(1 - r)$.

Как показано в работах [S6, S10], в традиционной вычислительной системе существует значительная вероятность того, что после выбора какого-то первого слова будут использоваться следующие за ним в памяти слова. Типичной является средняя длина такой цепочки в 3—10 слов. Параметр r позволяет параметрически учесть это обстоятельство в аналитической модели; значение r должно возрастать с увеличением размеров блока.

Для того чтобы понять и проанализировать функционирование буфера, удобно представить статистические количественные величины, характеризующие частоту использования слов или блоков (λ_i) с помощью следующих приближенных дискретных эмпирических выражений, упорядоченных по убыванию частоты использования:

$$\lambda_i = \alpha_1/[(\alpha_0 + iB)^{\alpha+1}],$$

где α, α_0 и α_1 — константы, а B — число слов в блоке.

[1] LRU (Least Referenced Unit) — наименее используемый блок — стратегия замещения блоков в буфере, в соответствии с которой блоки удаляются из буфера по признаку их наименьшего использования. FIFO (First-In — First-Out) — «Первым пришел — первым ушел» — стратегия, в соответствии с которой удаляется блок, дольше других находящийся в буфере. — *Прим. перев.*

Так как сумма вероятностей должна быть равна 1, можно записать

$$\sum_{i=1}^{m} \lambda_i = \sum_{i=1}^{m} \frac{\alpha_1}{(\alpha_0 + iB)^{\alpha+1}} = 1,$$

где m равно количеству блоков в памяти. При условии, что блоки не слишком велики и частота использования быстро снижается, дискретное распределение может быть в разумных пределах аппроксимировано непрерывным с неограниченным верхним пределом интегрирования, имеющим следующий вид:

$$\int_0^\infty \alpha \frac{a^\alpha}{(a + fW_B)^{\alpha+1}} W_B df = 1.$$

Безразмерная величина f представляет объединение слов памяти в блоки, по объему равные буферу. Подобная аппроксимация предполагает, что при нахождении параметров в разумных пределах снятие ограничений с верхней границы интегрирования влечет за собой пренебрежимо малую ошибку:

$$\int_{m/W_B}^\infty \alpha \frac{a^\alpha}{(a + fW_B)^{\alpha+1}} W_B\, df = 0.$$

Требование рационального соотношения параметров сводится к тому, что основная память должна иметь значительно бо́льший объем, чем кэш, в результате чего хвостовая часть графика плотности распределения имеет быстро убывающий характер. Если плотность распределения не резко убывает с увеличением f, необходимо прибегнуть к соответствующему верхнему пределу интегрирования. (Практика показывает, что значение показателя степени α лежит около 1 и слегка возрастает с увеличением объема блока.)

Чтобы проанализировать производительность буфера, снова предположим, что слова или блоки в памяти упорядочены на основе частоты их использования посредством следующего упрощенного закона распределения:

$$\lambda_i = \frac{a}{(a + iB)^2}, \qquad i = 1, \ldots, M,$$

где $a = a_0 = a_1$ (аналогичные результаты были бы получены и при использовании более точных законов распределения).

Вероятность того, что слово, занесенное в буфер, не окажется выбранным при следующем обращении к памяти, в предположении, что поток обращений носит независимый

характер, задается соотношением (параметр B для простоты принят равным 1)

$$P\,(\text{невыборки при следующем обращении}) = \left[1 - \frac{a}{(a+i)^2}\right].$$

При вероятности неудачного обращения M_r необходимо в среднем $1/M_r$ запросов к памяти, чтобы имело место неудачное обращение; для того чтобы слово было занесено в буфер, а затем возвращено из него в память, должно быть не более W_B неудачных обращений.

При движении блока по буферному LRU-стеку он может погружаться в более низкие уровни стека либо при выборе слова, либо при неудачных обращениях. Для того чтобы блок покинул буфер, необходимо в среднем меньше чем W_B/M_r обращений; введем параметр k для уточнения среднего числа обращений, необходимых для того, чтобы блок был удален из буфера.

Затем можно записать соотношение для вероятности того, что блок покинет буфер до того, как будет выбран:

$$P\left\{\begin{matrix}\text{новый блок в буфере не}\\ \text{выбирается при } W_B/M_rk\\ \text{обращениях}\end{matrix}\right\} = \left[1 - \frac{a}{(a+i)^2}\right]^{W_B/M_rk}.$$

С учетом того, что типичные значения a и i составляют

$$\frac{a}{(a+i)^2} \ll 1 \text{ и } (1-\varepsilon)^{1/\varepsilon} \approx 1/e,$$

можно записать

$$P\left\{\begin{matrix}\text{блок удаляется из}\\ \text{буфера за } W_B/M_rk\\ \text{обращений}\end{matrix}\right\} = \left(1 - \frac{a}{(a+i)^2}\right)\exp\left[\frac{(a+i)^2}{a}\cdot\frac{W_B}{M_rk}\,\frac{a}{(a+i)^2}\right]\times$$

$$\times\,(e)\exp\left[-\frac{W_B}{M_rk}\cdot\frac{a}{(a+i)^2}\right].$$

Записывая это выражение с использованием непрерывной переменной $f\,(i = fW_B)$, получаем

$$P\,\{\text{блок удаляется}\} = (e)\exp\left[-\frac{W_B}{M_rk}\cdot\frac{A}{(a+fW_B)^2}\right].$$

Вероятность того, что блок находится в буфере, определяется при этом соотношением

$$P_1\left\{\begin{matrix}\text{блок находится}\\ \text{в буфере}\end{matrix}\right\} = P_1 = 1 - (e)\exp\left[-\frac{W_B}{M_rk}\cdot\frac{A}{(a+fW_B)^2}\right].$$

Последнее выражение получено следующим образом. Пусть P_1 — вероятность того, что блок находится в буфере, а P_0 — вероятность того, что блока в буфере нет.

Рассмотрим приведенную ниже схему очереди:

{Вероятность того, что блок покинет кэш до того, как к нему вновь произойдет обращение}



{Скорость, с которой блоки продвигаются к верхушке стека}

$$\text{Вых. поток} = \frac{a}{(a+fW_B)^2} R(e) \exp\left[-\frac{W_B}{M_r k}\cdot\frac{a}{(a+fW_B)^2}\right]$$

(для всех блоков),

$$\text{Вх. поток} = P_0 \frac{a}{(a+fW_B)^2} R \text{ (для блоков, не находящихся в кэше)},$$

R — количество обращений с выбором блоков за секунду. Отметим, что $P_1 + P_0 = 1$; Вх. поток = Вых. поток; в состоянии равновесия

$$P_1 = 1 - P_0 = 1 - (e)\exp\left[-\frac{W_B}{M_r k}\cdot\frac{a}{(a+fW_B)^2}\right].$$

График P_1 представлен на рис. 4.1, а график частоты использования — на рис. 4.2.

Вероятность неудачного обращения M_r и параметр k можно оценить, принимая во внимание, что если вероятность пребывания блока в буфере распространяется на все блоки,

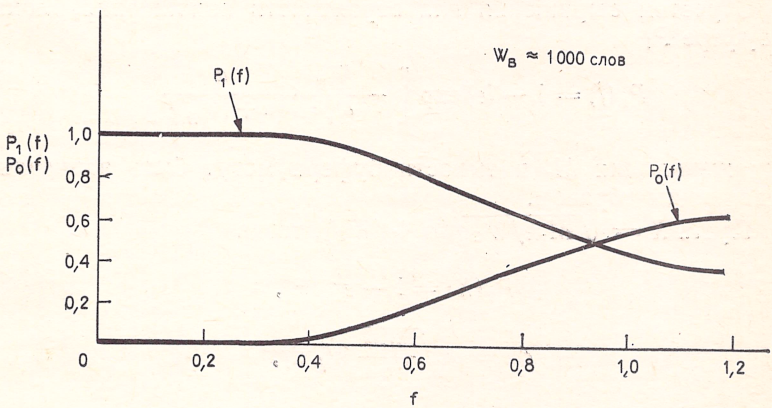


Рис. 4.1. Вероятность пребывания блоков в буфере и вне буфера как функция их упорядоченности по частоте использования: $[\lambda(f) = a/(a+fW_B)^2]$.

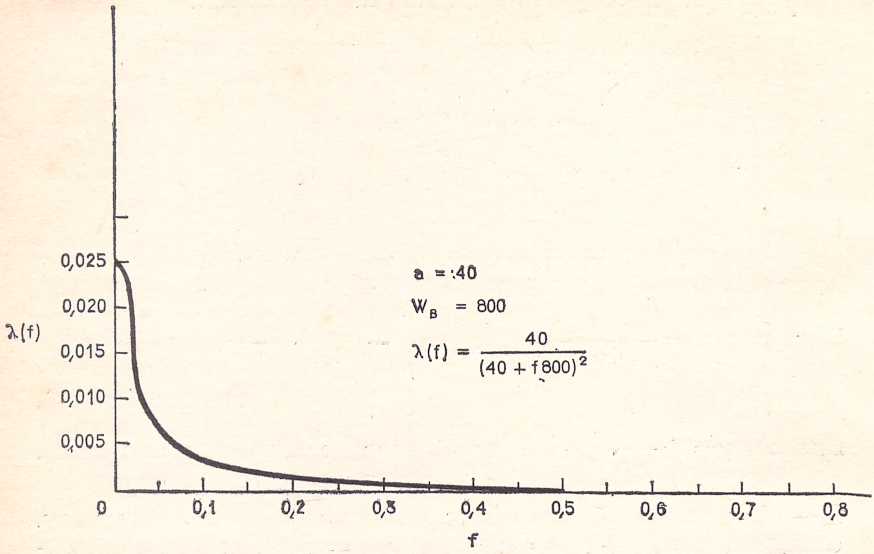


Рис. 4.2. Предполагаемое распределение частот использования.

то суммарная емкость резидентных блоков должна быть равна емкости буфера W_B; когда на все блоки распространяется произведение вероятности отсутствия блока в буфере на частоту использования, можно получить вероятность неудачного обращения. Располагая этими двумя результатами, наряду с вероятностью неудачного обращения можно определить параметр k.

С учетом выражения для вероятности нахождения блока в буфере P_1

$$P_1(f) = 1 - (e)\exp\left[-\frac{W_B}{M_r k}\cdot\frac{a}{(a+fW_B)^2}\right]$$

выражения для M_r и емкости буфера могут быть записаны следующим образом.

Пусть $X = \sqrt{M_r k/W_B a}\,(a + fW_B)$, тогда

$$P_1(X) = 1 - e^{-(1/x^2)},$$

$$dx = \sqrt{\frac{M_r k}{W_B a}}\, W_B\, df,$$

$$W_B = \sqrt{\frac{W_B a}{M_r k}} \int_{-\infty}^{\infty} \frac{(1 - e^{-1/x^2})}{\sqrt{M_r k a/W_B}}\, dx.$$

Приняв во внимание, что $\sqrt{M_r ka/W_B} \ll 1$, и выполнив численное интегрирование, получим

$$W_B = \sqrt{\frac{W_B a}{M_r k}}\left[1{,}783 - \sqrt{\frac{M_r ka}{W_B}}\right],$$

$$W_B + a = \sqrt{\frac{W_B a}{M_r k}}\, 1{,}783,$$

$$M_r = \frac{W_B}{(W_B + a)^2}\,\frac{a}{k}\, 1{,}783^2.$$

Вероятности удачного и неудачного обращений могут быть теперь непосредственно получены следующим образом:

$$H_r = \int_0^\infty \left(1 - (e)\exp\left[-\frac{W_B}{M_r k}\cdot\frac{a}{(a + fW_B)^2}\right]\right)\frac{aW_B\, df}{(a + fW_B)^2} =$$

$$= 1 - \int_0^\infty \frac{M_r k}{M_r k}(e)\exp\left(-\frac{W_B}{M_r k}\cdot\frac{a}{(a + fW_B)^2}\right)\frac{aW_B}{(a + fW_B)^2}\Big) df.$$

Принимая во внимание, что $H_r = 1 - M_r$, и полагая

$$u = \frac{1}{\sqrt{M_r k/W_B a}\,(a + fW_B)},$$

находим

$$M_r = \sqrt{\frac{M_r ka}{W_B}}\int_{-\infty}^{0} e^{-u^2}\, du.$$

Выполнив снова численное интегрирование и принимая во внимание, что $\sqrt{W_B/aM_r k} \gg 1$, получаем

$$M_r = \frac{ka}{W_B}\cdot\frac{\pi}{4}.$$

На основании этих двух выражений можно определить k:

$$M_r = \frac{ka}{W_B}\cdot\frac{\pi}{4},$$

$$M_r = \frac{W_B}{(a + W_B)^2}\cdot\frac{a}{k}\,(1{,}783)^2,$$

$$k^2 = \frac{W_B^2}{(a + W_B)^2}\cdot\frac{(1{,}783)^2 2^2}{\pi},$$

$$k^2 = \frac{W_B}{(a + W_B)}\cdot\frac{1{,}783 \times 2}{\sqrt{\pi}}.$$

Затем получаем простое выражение для вероятности неудачного обращения:

$$M_r = \frac{ka}{W_B} \cdot \frac{\pi}{4} = \frac{a}{W_B} \cdot \frac{\pi}{4} \cdot \frac{W_B}{(a + W_B)} \cdot \frac{1{,}783 \cdot 2}{\sqrt{\pi}} = \frac{a\,(1{,}58)}{(a + W_B)}.$$

Следует отметить, что существует большая вероятность пребывания первых W_B наиболее часто используемых слов в буфере. Используя очень простую модель стека, в соответствии с которой первые W_B ячеек стека пребывают в буфере [M8], можно записать

$$H_{r1} = \int_0^1 \frac{a_1 W_B\, df_1}{(a + f_1 W_B)^2},$$

$$M_{r1} = \int_1^\infty \frac{a_1 W_B\, df_1}{(a + f_1 W_B)^2} = \frac{a_1}{(a + W_B)} = \frac{a \times 1{,}58}{(a + W_B)},$$

где $a_1/(a + f_1 W_B)^2$ — вероятность выбора слова на расстоянии $f_1 W_B$ для занесения в стек.

Вследствие того что в буфере всегда находится несколько слов из числа менее используемых, вероятность неудачного обращения увеличивается в 1,58 раза по сравнению с ситуацией, когда слова упорядочены идеальным образом. Если бы частоты использования слов были известны заранее, то наиболее часто используемые слова всегда находились бы в буфере и вероятность неудачного обращения была бы в 1—1,58 раза меньше, чем в случае LRU-стека. Выполненный анализ показывает, что частота использования ячейки стека может быть получена на основании распределения частот использования блоков. Из-за того что время от времени в буфер попадают блоки, характеризующиеся низкими частотами использования, распределение частот в стеке несколько «растягивается» по сравнению с распределением частот использования. При повторном обращении к уже использованному слову вероятность неудачного обращения уменьшается в $(1 - r)$ раз.

В качестве рабочей модели вероятности неудачного обращения к буферу в дальнейших расчетах будет использоваться выражение

$$M_r = a_1/(a_0 + W_B)^\alpha.$$

Частота использования слов или блоков в соответствии с их положением в стеке задается как

$$Z(f) = \alpha a_1/(a_0 + f W_B)^{\alpha+1}.$$

Кроме того, как будет показано ниже, инкрементная вероятность неудачного обращения к буферу, заполненному jW_B словами, приближенно характеризуется следующим выражением:

$$M_r(j) = a_1/(a_0 + jW_B)^{\alpha}$$

в предположении, что имеет место приведенное ранее стационарное распределение частот использования слов. Если бы было применено другое выражение для частоты использования слов, были бы получены аналогичные результаты. отличающиеся лишь немного константами.

4.2. ЗАВИСИМОСТЬ ВЕРОЯТНОСТИ УДАЧНОГО ОБРАЩЕНИЯ ОТ РАЗМЕРОВ БЛОКА

Если ввести в рассмотрение возможность повторных обращений к блокам слов, находящимся в буфере, можно предсказать многие качественные характеристики зависимости вероятности удачного обращения от размеров блока.

Будем считать, что при наличии потоков команд и данных существует большая вероятность того, что следующее обращение происходит к очередному соседнему слову. Предполагая, что начальные запросы к новому блоку имеют произвольное распределение, а большинство запросов состоит в переходе к соседнему, предыдущему или следующему адресу, можно аппроксимировать длину укороченной цепочки слов блока следующим выражением:

$$l = \frac{1}{B} \sum_{J=1}^{B} \left\{ \sum_{n=1}^{J} \frac{n(L-1)^{n-1}}{L^n} + \frac{J(L-1)^J}{L^J} \right\},$$

где B — число слов в блоке, а L — средняя длина цепочки слов в памяти, не разделенной на блоки. Выражение предполагает одинаковую вероятность обращения к каждому слову блока и марковский характер процесса.

На рис. 4.3 сопоставлены кривые, отражающие изменения вероятности выбора очередного слова из того же блока, что и предыдущего, описываемые этой простой моделью и экспериментально полученные Шерром [S10]. Совпадение теоретических и экспериментальных данных оказывается достаточно близким при соответствующем выборе параметров.

В простой программе, как правило, можно выделить четыре типичные области адресации: команд, двух операндов и результатов. Неудивительно, что, если области размещения данных совмещаются с областью размещения команд, то ве-

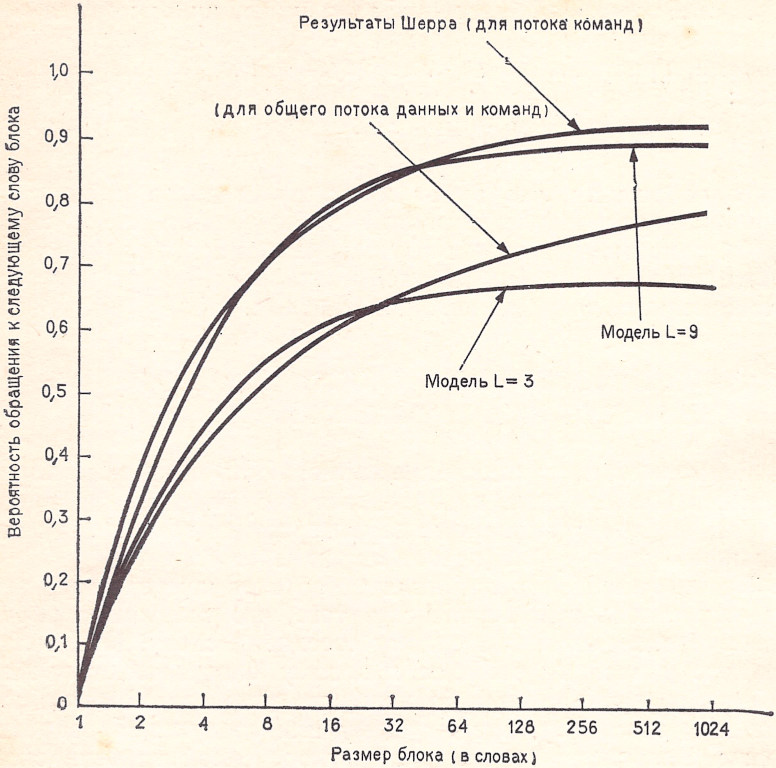


Рис. 4.3. Вероятность следующего обращения к тому же блоку.

роятность нахождения следующего слова в том же блоке, где находилось предыдущее, уменьшается. Что же касается функционирования буфера, то вероятность неудачного обращения может поддерживаться на низком уровне, пока очередное слово каждой из областей адресации оказывается в том же блоке, что и предыдущее. Эффективная длина цепочки слов в буфере определяется путем соответствующего усреднения длин цепочек каждого из полей.

Используя ранее определенные параметры, можно записать следующее соотношение для модифицированной модели независимых обращений:

$$1 - r = 1/l.$$

Если в буфер и из него перемещаются отдельные слова, то, очевидно, длина цепочки блоков оказывается весьма близкой к 1. Повторное обращение к тому же слову имеет место

только при выполнении столь необычной команды, как переход к той же ячейке.

В обычных программах вероятности удачных обращений резко убывают при уменьшении числа блоков до четырех и далее. Одновременно в активном состоянии пребывает не более четырех блоков: один блок программы, два блока операндов и один блок результатов, относящихся к одной программе. При введении зависимости вероятности неудачного обращения от числа блоков и размера блоков путем аппроксимации первого порядка с помощью размера блока B и эмпирического параметра δ получается следующее выражение для вероятности неудачного обращения:

$$M_r = \frac{a^{\alpha}(1{,}58)}{l\left[a + W_B\left[1 - \frac{\delta(B-l)}{W_B}\right]\right]^{\alpha}} = \frac{a^{\alpha}(1{,}58)}{l\,[a + W_B - (B-l)\,\delta]^{\alpha}}.$$

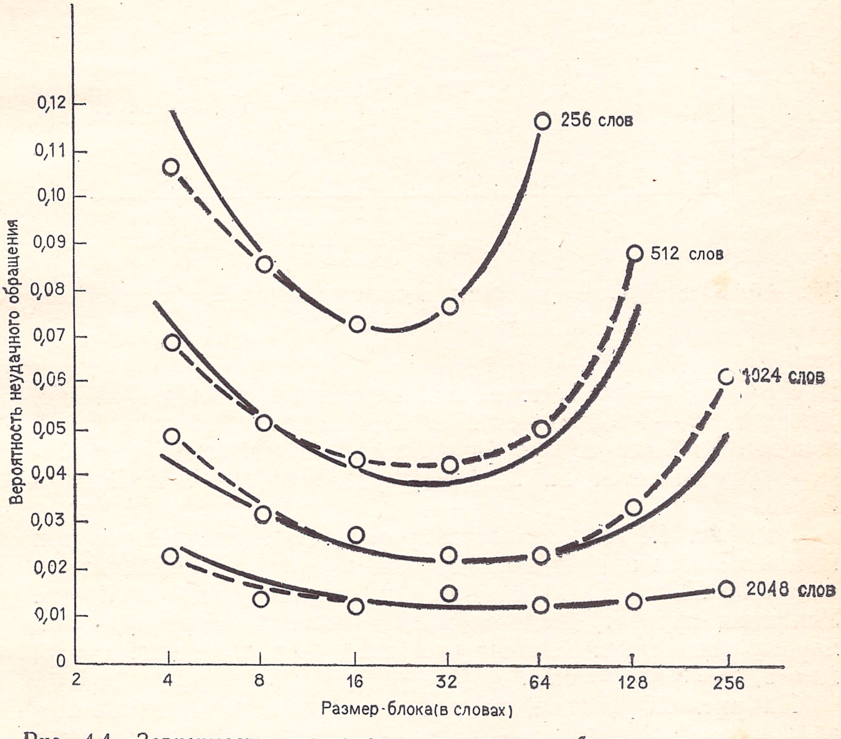


Рис. 4.4. Зависимость вероятности неудачного обращения от размеров кэша и блоков.

$$M_r = \frac{16}{l\,[23 + W_B - (B-l)\,3]^{0{,}75}}.$$

– – ○ – – экспериментальные данные Мэтсона [M10];
——— модель.

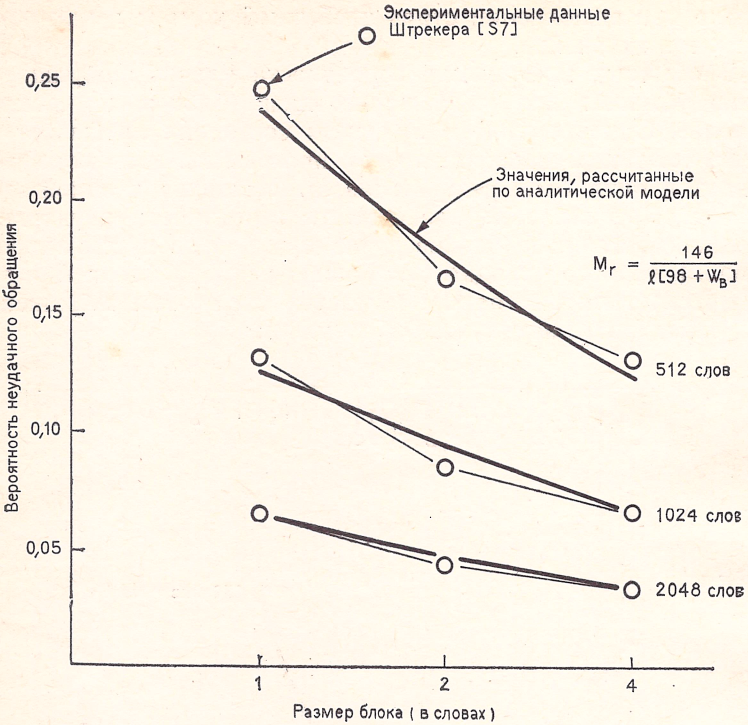


Рис. 4.5. Зависимость вероятности неудачного обращения от размеров кэша и блоков.

В случае $B = l = 1$, т. е. при пересылке одиночных слов, разность $B - l$ обращается в 0. Отметим, что если $\delta = 4$ и в памяти было четыре блока, то эффективная емкость буфера уменьшается на величину разности $B - l$ в каждом блоке.

На рис. 4.4 сопоставлены результаты, полученные при использовании аналитической модели, с экспериментальными данными Мэтсона [M10] при увеличении размеров блока. Учитывая простоту модели, можно считать, что результаты хорошо совпадают. Величины параметров составляют: $a = 23$, $L = 5$, $\delta = 3$ и $\alpha = 0{,}75$.

На рис. 4.5 данные, полученные при использовании модели, сопоставлены с экспериментальными результатами Штрекера [S7] для небольших размеров блоков. Совпадение результатов при рациональном выборе параметров оказывается опять хорошим.

4.3. ЗАПОЛНЕНИЕ БУФЕРА И ВЕРОЯТНОСТЬ НЕУДАЧНОГО ОБРАЩЕНИЯ В НАЧАЛЕ ВЫПОЛНЕНИЯ ЗАДАНИЯ

Для целей анализа предположим, что в начале выполнения нового задания буфер пуст. Каплан и Уиндер [К2] показали, что в типичном случае лишь 20 % слов, связанных с предыдущим заданием, остаются в буфере, когда процессор переключается на выполнение нового задания. Эти слова скорее всего относятся к операционной системе. Снова будем считать, что f — часть буфера, заполненная словами, относящимися к новому заданию, а вероятность неудачного обращения задается соотношением

$$M_r = a_1/(a_0 + fW_B)^\alpha.$$

Пусть R — скорость запросов памяти (количество запросов в секунду), а t — время (в секундах) с начала выполнения нового задания. Пусть, кроме того, B — количество слов в блоке. Заполнение буфера может тогда быть описано (при использовании стратегии LRU или FIFO с помощью следующего дифференциального уравнения:

$$df' = \frac{1}{W_B}\,\frac{a_1}{(a_0 + W_B f')^\alpha}\,B \cdot R\,dt'.$$

Это выражение предполагает, что перед занесением в буфер слов, относящихся к новому заданию, из него удаляются все старые слова. Разделяя переменные и интегрируя по частям, получаем

$$\int\limits_0^{f(t)} W_B\,(a_0 + W_B f')^\alpha\,df' = \int\limits_0^t a_1 BR\,dt',$$

$$\left(a_0 + W_B f(t)\right)^{\alpha+1} - a_0^{\alpha+1} = (\alpha + 1)\,BRta_1,$$

$$f(t) = \frac{1}{W_B}\left[(\alpha + 1)\,a_1 BRt + a_0^{\alpha+1}\right]\frac{a_0}{W_B}.$$

Полное число запросов памяти, необходимое для заполнения буфера, можно легко найти, положив $f = 1$:

$$RT_{fL} = \frac{\left(a_0 + W_B\right)^{\alpha+1} - a_0^{\alpha+1}}{(\alpha + 1)\,a_1 B}.$$

Время заполнения T_{fL} определяется как

$$T_{fL} = \frac{\left(a_0 + W_B\right)^{\alpha+1} a_0^{\alpha+1}}{(\alpha + 1)\,a_1 BR}.$$

Среднюю вероятность удачного обращения (M_{rf}) в процессе заполнения можно рассчитать непосредственно по соотношению

$$\frac{a_1}{(a_0+f(t)W_B)^\alpha}=\frac{a_1}{\left[(1/W_B)\left(a_0^{\alpha+1}+(\alpha+1)a_1BRt\right)^{\alpha/\alpha+1}\cdot\frac{a_0}{W_B}\right]^\alpha},$$

пренебрегая величиной a_0W_B.

$$M_{rf}\approx\frac{1}{BRT}\int_0^T\frac{a_1BR\,dt}{\left(a_0^{\alpha+1}+(\alpha+1)a_1BRt\right)^{\alpha/\alpha+1}}\text{ при } T\leqslant T_{fL}.$$

Интегрируя, получаем

$$M_{rf}=\frac{a_0}{BRT}\left[\left(1+(\alpha+1)\frac{a_1}{a_0^\alpha}\frac{BRT}{a_c}\right)^{1/\alpha+1}-1\right].$$

Введем обозначение $u=BRT/a_0$; тогда

$$M_{rf}=\frac{1}{u}\left[\left(1+\frac{(\alpha+1)a_1}{a_0^\alpha}u\right)^{1/\alpha+1}-1\right].$$

После заполнения инкрементная вероятность удачного обращения принимает свое нормальное значение. Усредняя по времени для $T>T_{fL}$, получаем

$$M_{re}=\frac{1}{T}\left[T_{fL}M_{rf}(T_{fL})+(T-T_{fL})\frac{a_1}{(a_0+W_B)^\alpha}\right].$$

На рис. 4.6 ($a_e=a_1=40$) приведены графики изменения инкрементной и средней вероятностей удачного обращения при заполнении буфера емкостью 2000 слов.

Если используется стратегия произвольного замещения, то перед окончательным заполнением буфера новые слова постоянно заменяются; дифференциальное выражение в этом случае имеет вид

$$df'=\frac{1}{W_B}\left[\frac{a_1}{(a_0+f'W_B)^\alpha}-\frac{a_1}{(a_0+W_B)^\alpha}\right]BR\,dt'.$$

Разделяя переменные и применяя непосредственное интегрирование, при $\alpha=1$ получаем

$$f(t)=1-(e)\exp\left\{-\frac{W_B}{a+W_B}f(t)\right\}(e)\exp\left[-\frac{a_1BRt}{(a_0+W_B)^2}\right].$$

Заполнение буфера происходит в этом случае медленнее.

На рис. 4.7 представлены графики изменения вероятности удачного обращения ($a_0=a_1=40$) при заполнении буферов различных размеров с использованием стратегии произволь-

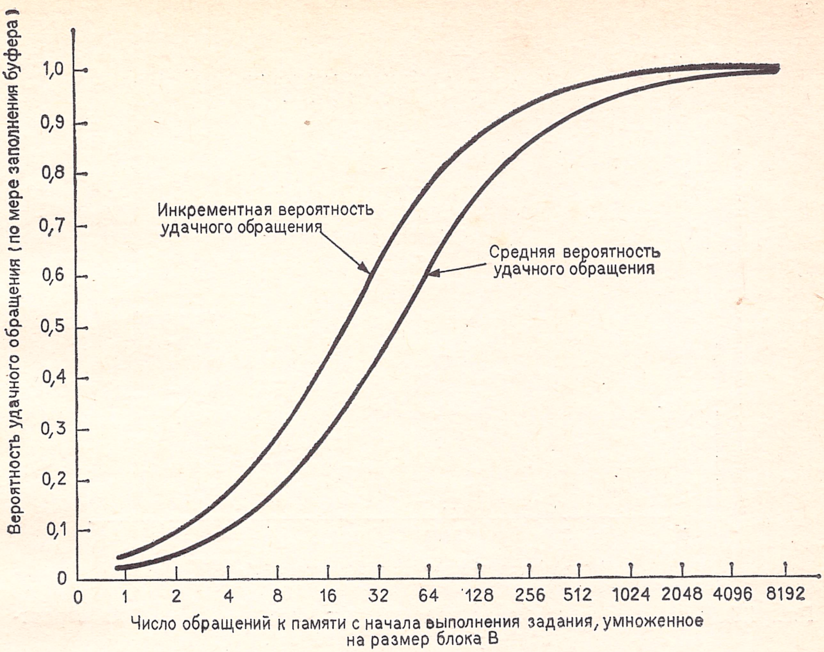


Рис. 4.6. Зависимость вероятности удачного обращения от заполнения буфера. (Стратегия замещения LRU.)

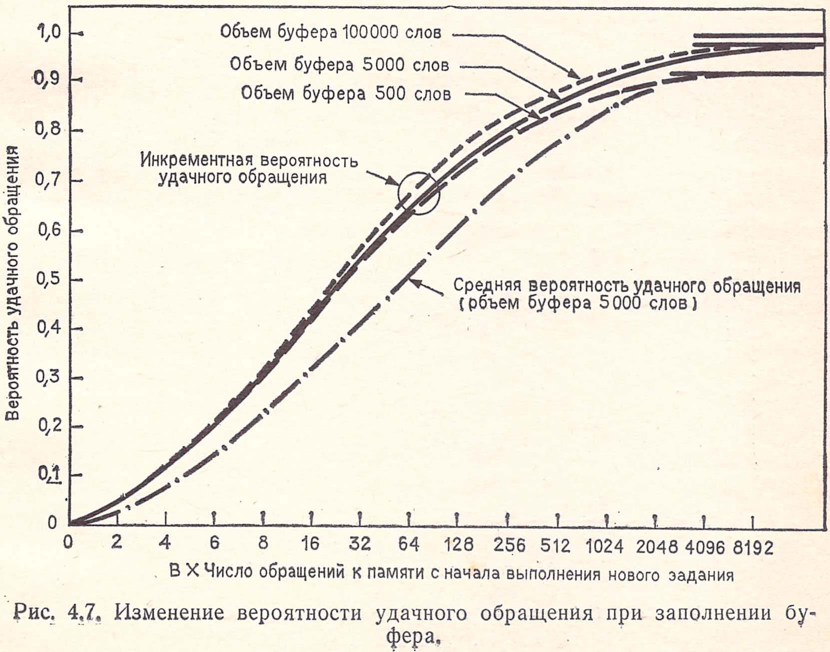


Рис. 4.7. Изменение вероятности удачного обращения при заполнении буфера.



Рис. 4.8. Влияние переключения заданий, т. е. очистки буфера, на вероятность удачного обращения.

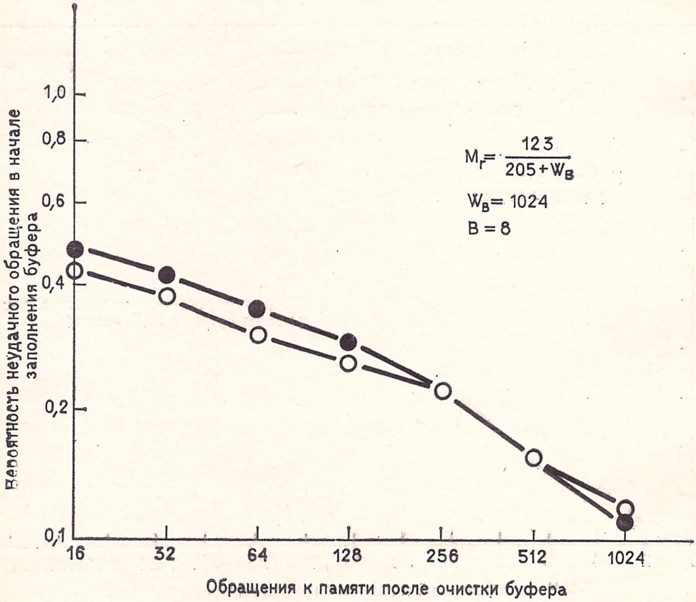


Рис. 4.9. Обращения к памяти после очистки буфера.
● модель; ○ данные Истона [E2].

ного замещения. Там же приведен график изменения средней вероятности удачного обращения в зависимости от времени, помноженного на число обращений, имевших место с начала выполнения задания.

Эти графики отчетливо показывают, что, если размер блока составляет одно слово, то во всех случаях необходимо не менее тысячи обращений, чтобы вероятность удачного обращения превысила 0,9. Очевидно, что при реализации заданий интерактивного характера, когда до того, как произойдет переключение к новому заданию, выполняется несколько тысяч команд, целесообразно использовать большие блоки памяти, чтобы заполнение буфера происходило с меньшим числом неудачных обращений и достигалось бóльшее значение средней вероятности удачного обращения.

На рис. 4.8 сопоставлены графики изменения вероятности неудачного обращения, построенные в соответствии с простой теорией и по результатам измерений, проведенных Штрекером [S7] при небольших размерах блоков. Параметр α был положен равным 1. Совпадение двух видов результатов опять оказывается хорошим. Для заполнения буфера необходимо около 4900 обращений к памяти. На рис. 4.9 сопоставлены экспериментальные результаты Истона [E2] и расчетные данные в соответствии с простой теорией при $\alpha = 1$. Совпадение и здесь вполне удовлетворительное. Оно было бы еще точнее при более тщательном выборе α. В этом случае для заполнения буфера потребовалось бы примерно 800 обращений к памяти. Необходимое число запросов уменьшается вследствие увеличения размеров блоков до 8 слов и постоянных a_0 и a_1. Достигаемое при этом повышение производительности невелико из-за возрастания временной задержки на пересылку блока в случае больших размеров последнего. Кроме того, очевидно, что, если размеры блоков оказываются слишком большими, в буфер заносится много слов, не входящих в число слов, используемых наиболее часто.

УПРАЖНЕНИЯ

1. Считая, что $\lambda(f) = \nu W_B e^{-fW_B}$, где $\nu = 10^{-2}$, а $W_B = 1000$, определите вероятность того, что некоторое слово находится в кэше, как функцию от f.

2. Анализ, в ходе которого рассматривались частота использования блоков и вероятность удачного обращения, опирался на предположение об использовании LRU-стека, где замещению подлежит блок, использовавшийся последним. Определите, каким образом скажется на вероятности удачного обращения применение алгоритма FIFO.

3. Считая, что частота использования подчиняется, как

в упр. 1, экспоненциальному закону, выведите выражения для вероятностей неудачного и удачного обращения в виде функций от W_B.

4. Определите эффективную длину цепочки l при объеме блока 16 слов ($B = 16$) и длине цепочки 10 ($L = 10$).

5. Выражение для вероятности неудачного обращения при полном буфере имеет вид $50/(50 + 1024)$, причем объем блока равен 4. Определите число запросов памяти, необходимое для заполнения буфера.

6. Пусть выражение для вероятности неудачного обращения, приведенное в упр. 5, характеризует выполнение интерактивного задания, в ходе которого производится 5000 запросов памяти, после чего происходит переключение задания. Определите среднюю вероятность неудачного обращения.

7. При использовании LRU-стратегии замещения достигается меньшая вероятность неудачного обращения, чем при произвольном замещении. Сравните инкрементные вероятности неудачного обращения для этих стратегий после 1024 обращений, используя параметры буфера, приведенные в упр. 5.

8. Выполните упр. 7, увеличив размер блока до 32 слов ($B = 32$).

Глава 5

Иерархическая организация памяти

5.1. ВВЕДЕНИЕ

Хотя в небольших системах один кэш, как правило, служит для обработки слов, данных и команд, очевидно, что для повышения производительности систем памяти можно использовать несколько кэшей. Распределение этих кэшей может быть самым различным: по выполняемым функциям, модулям памяти или процессорам. Кроме того, очевидно, что иерархическая организация систем с кэш-памятью может содержать не только два, а три или более уровней. Ниже будут рассмотрены некоторые из видов подобных иерархических организаций.

5.2. БУФЕРИЗАЦИЯ МНОЖЕСТВА МОДУЛЕЙ ПАМЯТИ

Расширение возможностей систем памяти в больших ЭВМ за счет совмещения обращений к модулям памяти является традиционным. Аналогичный подход можно реализовать в системах памяти с буфером путем буферизации отдельных модулей, которые работают с совмещением. Применение данного метода оказывается уместным, когда либо буфер имеет недостаточно высокое быстродействие, либо небольшая длительность программ препятствует достижению большой вероятности удачного обращения за счет использования большого буфера.

Чтобы проиллюстрировать эффективность данного метода, рассмотрим две возможные организации систем памяти, в одной из которых используется один большой буфер, а в другой каждому из работающих с перекрытием модулей сопоставлен отдельный буфер меньших размеров. Эти два вида организации памяти иллюстрируются на рис. 5.1 и 5.2.

Для того чтобы наиболее эффективно использовать четыре модуля при наличии одного общего буфера, было бы целесообразно иметь дело с блоками, состоящими из четырех слов. Применение блоков, содержащих по одному слову, в случае буферизации отдельных модулей представляло бы собой наиболее неблагоприятный случай. Однако из-за малых размеров блока для заполнения буфера тех же размеров потребовалось бы большее количество неудачных обращений. Чтобы учесть это обстоятельство, прибегнем к сле-

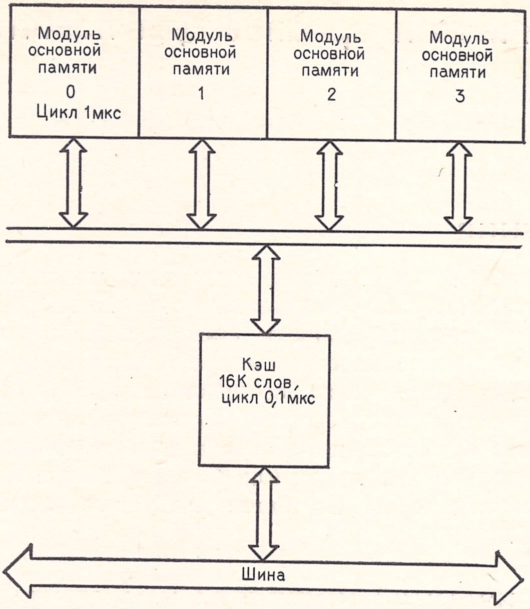


Рис. 5.1. Схема с одним буфером, обслуживающим несколько модулей.

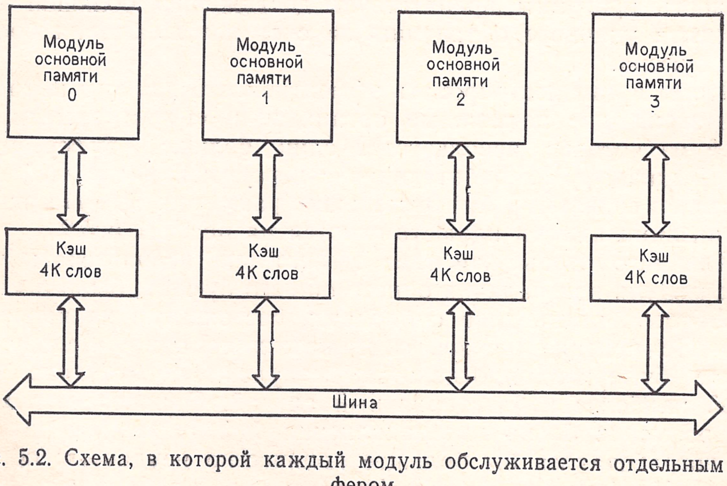


Рис. 5.2. Схема, в которой каждый модуль обслуживается отдельным буфером.

дующим приближенным выражениям для вероятности неудачного обращения, которые соответствуют экспериментальным данным Штрекера [S7].

Для случая одного большого буфера $M_{rs} = 20/(20 + W_B)$.

В случае множества буферов $M_{rm} = 40/(40 + W_B)$, где W_B — количество слов в буфере.

Чтобы упростить расчеты, предположим, что в обеих системах используется алгоритм флагового свопинга, причем доля блоков, отмеченных флагами, составляет 0,2 и перекрытия отсутствуют. Применение алгоритмов с перекрытием еще более повышает производительность многомодульной системы памяти. Будем также считать, что количество одновременно занятых модулей определяется как $N^{0,56}$, где N — общее число модулей [H3], что соответствует предположению о независимости запросов памяти. Для большинства реальных программных потоков подобное предположение выглядит как пессимистичное.

Эффективное время цикла для одно- и многомодульного вариантов может быть достаточно просто рассчитано в предположении, что в системах обоих типов выполняется единственное и длинное задание.

В случае одного буфера

$$M_{rs} = \frac{20}{20 + 16\,384} = 0{,}00122,$$

$$T_{\text{с эфф}} = (1 - 0{,}00122)\,0{,}1 \text{ мкс} + 0{,}00122\,(0{,}1 + 1{,}2 \times 1{,}0) = 0{,}1015 \text{ мкс}$$

В случае множества буферов

$$M_{rm} = \frac{40}{40 + 4096} = 0{,}00967,$$

$$T_{\text{с эфф}} = (1 - 0{,}00967)\,0{,}1 + 0{,}00967\,(0{,}1 + 1{,}2 \times 1{,}0) = 0{,}1116 \text{ мкс.}$$

При работе модулей с перекрытием, когда $4^{0,56}$ из них постоянно поддерживаются в занятом состоянии, эффективное время цикла составляет $T_{\text{с эфф}}/4^{0,56} = 0{,}0513$ мкс.

В больших вычислительных системах применяется мультипрограммирование, в результате чего буферное пространство, отводимое каждому модулю, уменьшается. Рассмотрим ситуацию, когда в системе одновременно находится восемь заданий, каждому из которых произвольным образом отведена одна восьмая часть буфера, имеющего указанный выше объем. Как будет показано ниже, эффективное время цикла при этом возрастает из-за увеличения вероятности неудачного обращения; однако производительность многомодульного варианта организации продолжает и в этом случае оставаться более высокой, чем одномодульного. Чтобы в еще большей степени улучшить характеристики многомодульного варианта,

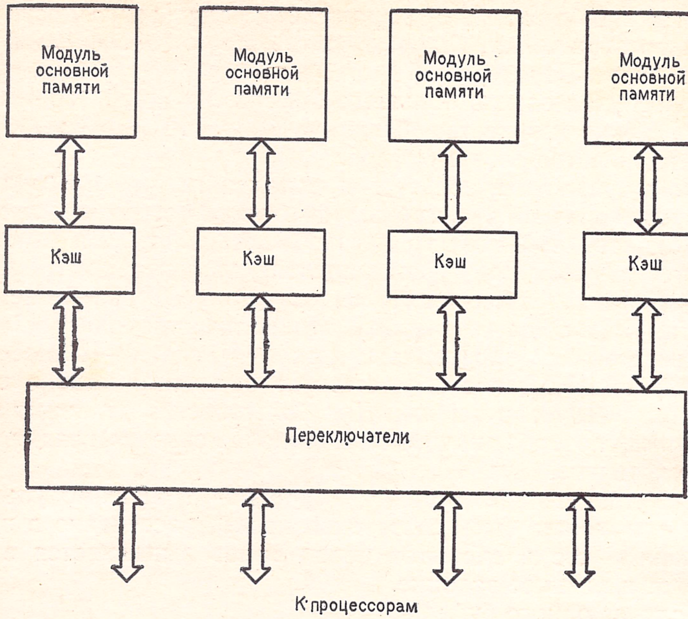


Рис. 5.3. Мультипроцессорная система с множеством кэшей.

можно увеличить размер буфера, что оказывает лишь небольшое улучшающее действие в одномодульном варианте.

В случае использования одного буфера

$$M_{rs} = \frac{20}{20 + 2048} = 0{,}00967,$$

$$T_{\text{с эфф}} = (1 - 0{,}00967)0{,}1 + 0{,}000967(0{,}1 + 1{,}2 \times 1{,}0) = 0{,}1115 \text{ мкс.}$$

Для многобуферного случая

$$M_{rm} = \frac{40}{40 + 512} = 0{,}0725,$$

$$T_{\text{с эфф}} = \frac{1}{4^{0,56}} [(1 - 0{,}0725)\, 0{,}1 + 0{,}0725\, (0{,}1 + 1{,}2 \times 1{,}0)] = 0{,}086 \text{ мкс.}$$

В общем случае система памяти с буфером может быть спроектирована таким образом, что ее эффективная производительность оказывается довольно высокой за счет совместного применения буферизации и временного совмещения; это приобретает особую важность, когда производительность отдельного буфера становится ограничивающим фактором.

Если кэш-память размещена ближе к модулям основной памяти, чем к процессору, то задержки, вносимые шиной, могут существенно ограничить производительность системы.

Рис. 5.3 иллюстрирует случай, когда кэш-память расположена близко к переключателю, обслуживающему мультипроцессоры. В этом случае процессоры по необходимости располагаются в непосредственной близости от переключателя, чтобы снизить шинные задержки, и образуют тесно связанную систему. Модули основной памяти могут быть в этом случае связаны с соответствующими кэшами более длинными шинами, так как требования к пропускной способности шин здесь ниже.

Для оптимального построения подобной системы желательно иметь много кэшей, функционирующих одновременно. Способ, которым организованы блоки в каждом из кэшей, зависит от вида выполняемой обработки информации. Если множество процессоров заняты решением независимых задач, блоки в кэше организуются обычным образом. Если же мультипроцессоры используются для конвейерной обработки векторных данных, то в оптимальном случае каждый кэш-блок должен содержать не соседние, следующие друг за другом слова, а лишь n-е по порядку слова, где n — число кэшей. Тогда последовательность слов будет поступать по очереди из всех кэшей. Каждый кэш в такой системе обслуживает пространство памяти, отдельное от зоны обслуживания всех других кэшей.

5.3. СВЯЗНОСТЬ ИНФОРМАЦИИ В КЭШ-ПАМЯТИ

На рис. 5.4 показана организация мультипроцессорной системы, в которой для каждого процессора имеется свой отдельный кэш. В подобных системах существует проблема сохранения связности информации. Схема организации памяти называется *связной*, если значение, получаемое из памяти при операции чтения, всегда является значением, занесенным в память при последней операции записи. Если в памяти хранится множество копий некоторой информации, а средств для одновременного обновления всех этих копий при записи не предусмотрено, то такая система памяти, очевидно, связной не является.

Классическое решение этой проблемы состоит в том, что для каждого кэша используется алгоритм сквозной записи. Кроме того, каждый кэш подключен к вспомогательному информационному тракту, по которому все другие активные устройства посылают адреса блоков, которые должны подвергнуться изменениям. Каждый кэш следит за информацией, поступающей по этому тракту, выявляя случаи совпадения передаваемых по нему номеров блоков с номерами, содержащимися в его собственном справочнике. При обнаружении совпадения содержимое блока должно быть обновлено или

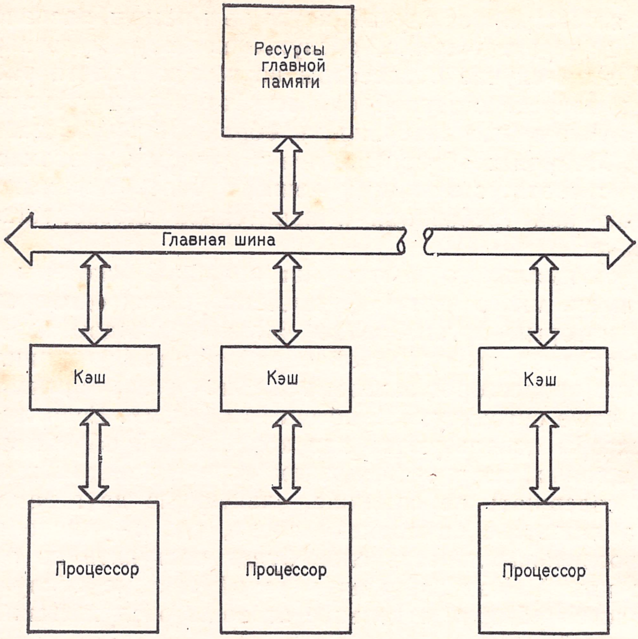


Рис. 5.4. Мультипроцессорная система с отдельным кэшем для каждого процессора.

отмечено как недостоверное. Ясно, что эти поисковые операции снижают эффективное быстродействие кэша.

Тэнг [Т3] предложил метод решения проблемы связности путем ведения и постоянного обновления копий справочников всех кэшей в основной памяти. Сенсье и Фотрие [С6] предложили метод, базирующийся на использовании для поддержания связности флагов трех типов и более эффективных алгоритмов свопинга. В соответствии с этим методом для каждого блока предусматривается свой «персональный» флаг. Если этот флаг в кэше установлен, это означает, что данный кэш является единственным местом, где находится достоверная копия блока, и может участвовать в свопинге. Флаги второго типа — это флаги присутствия в кэшах; ими снабжен каждый блок, находящийся в основной памяти, для указания, в каких кэшах находятся достоверные копии блока. Копии в кэшах могут совпадать, а могут и не совпадать с копиями, находящимися в основной памяти. Когда какой-то кэш обращается с запросом в основную память, указание считать информацию, находящуюся в нем, недостоверной и обновить ее должно быть послано только в те кэши, флаги присутствия которых находятся в установленном состоянии.

Флаги третьего типа представляют собой флаги изменений; они имеются у каждого блока в основной памяти. Такой флаг имеет нулевое значение, когда содержимое блока, находящегося в основной памяти, совпадает с копиями блока в кэшах, и устанавливается в 1, когда имеются различия. Флаги присутствия отражают физическое состояние системы памяти, потому что имеются в каждом кэше.

5.4. ОРГАНИЗАЦИЯ МУЛЬТИКЭШ-ПРОЦЕССОРНОЙ СИСТЕМЫ

Во многих случаях применения вычислительных систем желательно, чтобы они не только включали мультипроцессоры, но и допускали возможность добавления (или исключения) процессоров без необходимости внесения изменений в аппаратную часть системы памяти. Кроме того, часто оказывается целесообразным оптимизировать использование именно главной памяти и шины, а не отдельных процессоров. В этом разделе будет рассмотрен подход к организации кэш-памяти, который при незначительном усложнении аппаратуры кэшей существенно уменьшает шинный трафик по сравнению с простым методом сквозной записи [B4] и позволяет строить системы, размеры которых легко увеличивать и уменьшать. Подобные системы обеспечивают эффективный компромисс между снижением шинного трафика и увеличением сложности системы. Значительное повышение производительности может быть достигнуто в них, особенно по мере увеличения быстродействия процессоров, за счет наличия кэш-памяти у каждого процессора. Шинный график (или потоки информации, поступающие в распределенную память и выходящие из нее) может играть роль факторов, ограничивающих полную производительность систем такого рода.

Локальные шины типа Unibus[1]) имеют скорость трафика около $2{,}5 \times 10^6$ слов/с. Во многих современных мини-ЭВМ требуется, чтобы примерно с такой скоростью осуществлялась передача слов команд и данных, и если не применять кэш-память, то один процессор может полностью использовать всю пропускную способность шины. Введение кэша позволяет значительно снизить требования к шинному трафику со стороны процессора и делает возможным применение множества процессоров. Производительность вычислительной системы может быть увеличена при этом путем добавления процессоров до тех пор, пока снова не будет исчерпана пропускная способность шины. Если локальная шина или сеть связи имеет бóльшую протяженность, чем шина Unibus, и меньшую

[1]) Название, введенное фирмой Digital Equipment Corp.

скорость передачи слов, то применение кэша оказывается необходимым даже для обеспечения загрузки единственного процессора.

Анализ большого числа программ показывает, что в среднем 0,1—0,3 запросов памяти со стороны процессора приходится на операции записи [С6]. Таким образом, шинный трафик, источником которого является процессор, не менее чем на 10—30 % определяется запросами памяти, если для работы с кэшем используется алгоритм сквозной записи. Если же для операций с кэшем применяется алгоритм свопинга с флагами [Р3], то скорость шинного обмена может быть снижена, однако в этом случае возникают проблемы обеспечения связности памяти.

На рис. 5.5 представлена структура одношинной вычислительной системы, в которой часть памяти, включая главную память, распределена по различным процессорам. Из соображений простоты показанная на рисунке схема управления шиной представляет собой устройство обработки запросов шины и выдачи разрешений на ее использование. Предполагается, что кэш-память служит для работы как с данными, так и с командами.

Для обновления содержимого кэшей в случае записи множества копий используется система прерываний, основу которой составляют линии поиска источника прерываний и подтверждения его нахождения, показанные на рис. 5.5. На рис. 5.6 изображены временные диаграммы прохождения сигналов прерывания и подтверждения. В данной вычислительной системе поиск кэша, инициируемый шиной, имеет место только при выделяемых операциях записи, на которые указывает сигнал источника прерывания. (Эти операции записи в дальнейшем будут рассмотрены более подробно.) Из временных диаграмм на рис. 5.6 видно, что операции с кэш-памятью протекают гораздо быстрее, чем операции с основной памятью, и могут совмещаться во времени с записью в основную память. При этом операции поиска кэша не удлиняют промежутков времени, в течение которых шина используется для операций записи. Отдельный кэш увеличивает длительность реакции на запрос прерывания на величину, не превышающую один его собственный цикл.

В примере, показанном на рис. 5.6, подразумевается, что в мультипроцессорной системе передаче подлежит одно большое слово. В большинстве случаев блоки состоят из множества слов. Чтобы такой блок мог быть записан за один период синхронизации, необходимо предусматривать соответствующее тактирование для последовательной записи слов блока или увеличивать (в разумных пределах) ширину шины в соответствии с размерами блока.

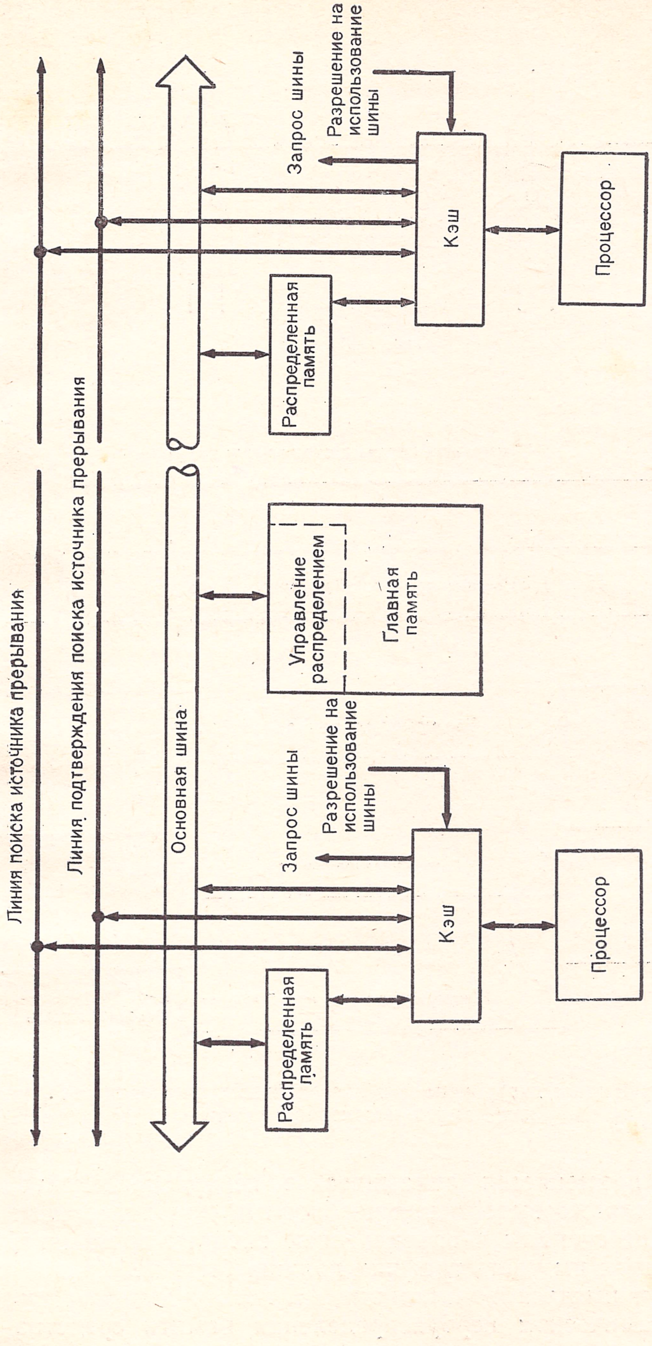


Рис. 5.5. Организация мультипроцессорной системы.

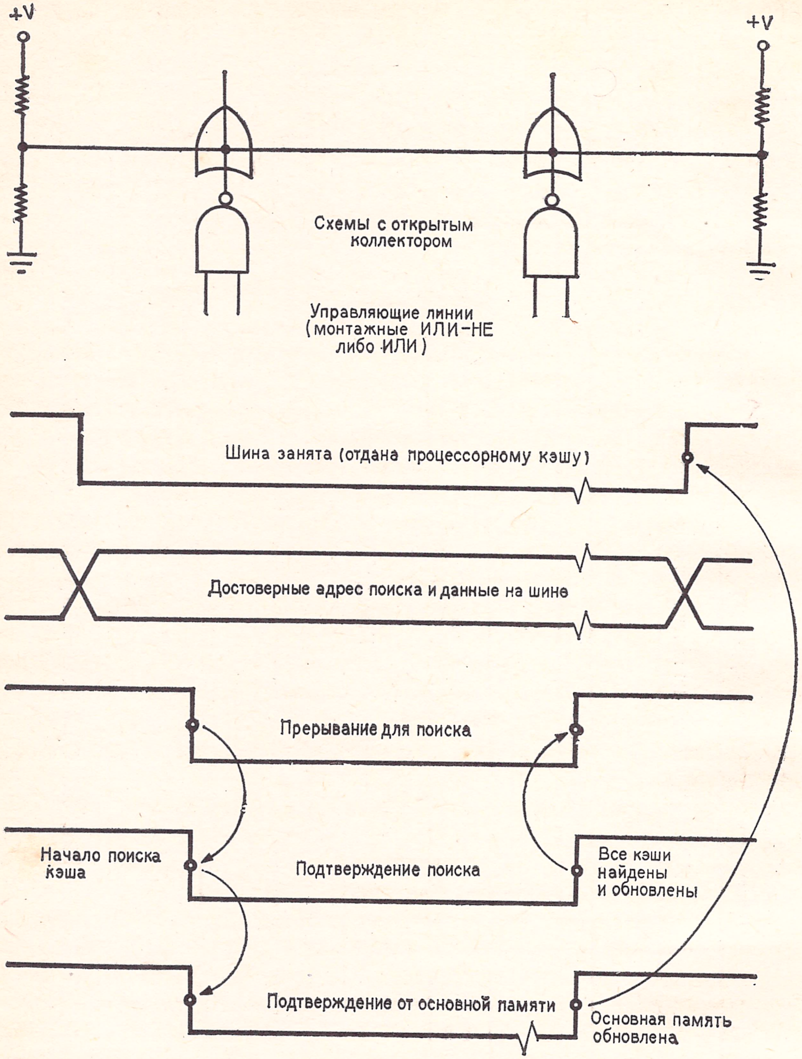


Рис. 5.6. Временные диаграммы при многократном изменении содержимого кэш-памяти и основной памяти.

Если применяется динамическое перераспределение памяти, то для его выполнения необходимы дополнительные управляющие сигналы и определенные затраты времени. Соответствующая им часть трафика должна быть небольшой и не будет приниматься во внимание в последующих расчетах. При динамическом перераспределении памяти средства уп-

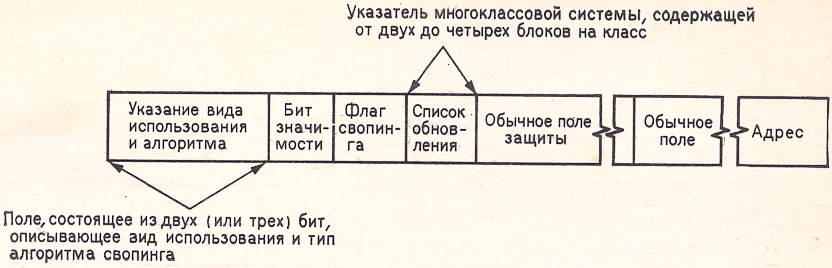


Рис. 5.7. Поля указателя кэша.

равления отведением в использование шины относятся к ресурсам главной памяти.

С точки зрения использования память, которой пользуется любой процессор системы, может быть подразделена на следующие виды:

1. Память, находящаяся там же, где процессор, и используемая исключительно этим процессором (*локальная*). К ней не могут обращаться другие процессоры, она не имеет отображения в центральной памяти.

2. Часть главной памяти, отведенная одному процессору (*собственная*). Это фиксированная часть главной памяти, запись и чтение содержимого которой осуществляется только одним процессором. Такая часть главной памяти постоянно занята одним процессором.

3. Распределенная или процессорная память, допускающая *только чтение* (*постоянная память*).

4. Часть главной памяти, которая временно отведена одному процессору (*занятая*). К этой части памяти имеет доступ только один процессор, пока у него не отпадает в этом надобность (при этом она становится *свободной*).

5. Память, используемая множеством процессоров и допускающая как чтение, так и запись.

Сущность метода заключается в том, что к полю защиты памяти каждого блока добавляются два бита (а если принимается во внимание вариант с локальной памятью, то три). Эта информация загружается в указатель кэша для обозначения типа каждого блока, находящегося в кэше. Кэш использует алгоритм сквозной записи только для распределенной памяти и алгоритм свопинга с флагами для всех других категорий памяти, если речь не идет о множественном копировании кэшей.

На рис. 5.7 иллюстрируется назначение двух (или трех) бит, которые должны быть добавлены к стандартному слову указателя при реализации рассматриваемого метода. Центральное устройство управления присваивает этим битам со-

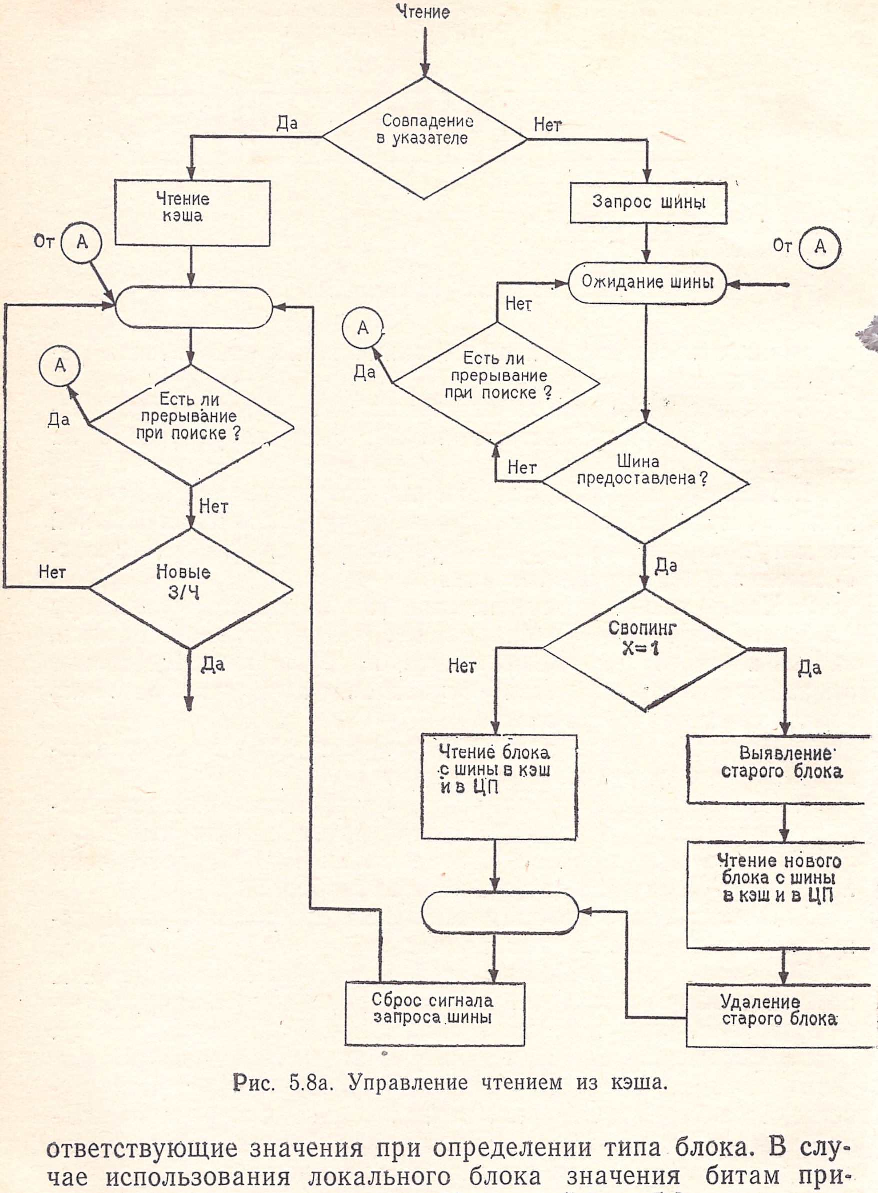


Рис. 5.8а. Управление чтением из кэша.

ответствующие значения при определении типа блока. В случае использования локального блока значения битам присваивает локальная память. На рис. 5.8а — 5.8в приведены схемы алгоритмов функционирования отдельных кэшей. Централизованные средства распределения памяти могут либо располагать программными моделями процессоров, использующих шину, либо получать от процессоров специфичные

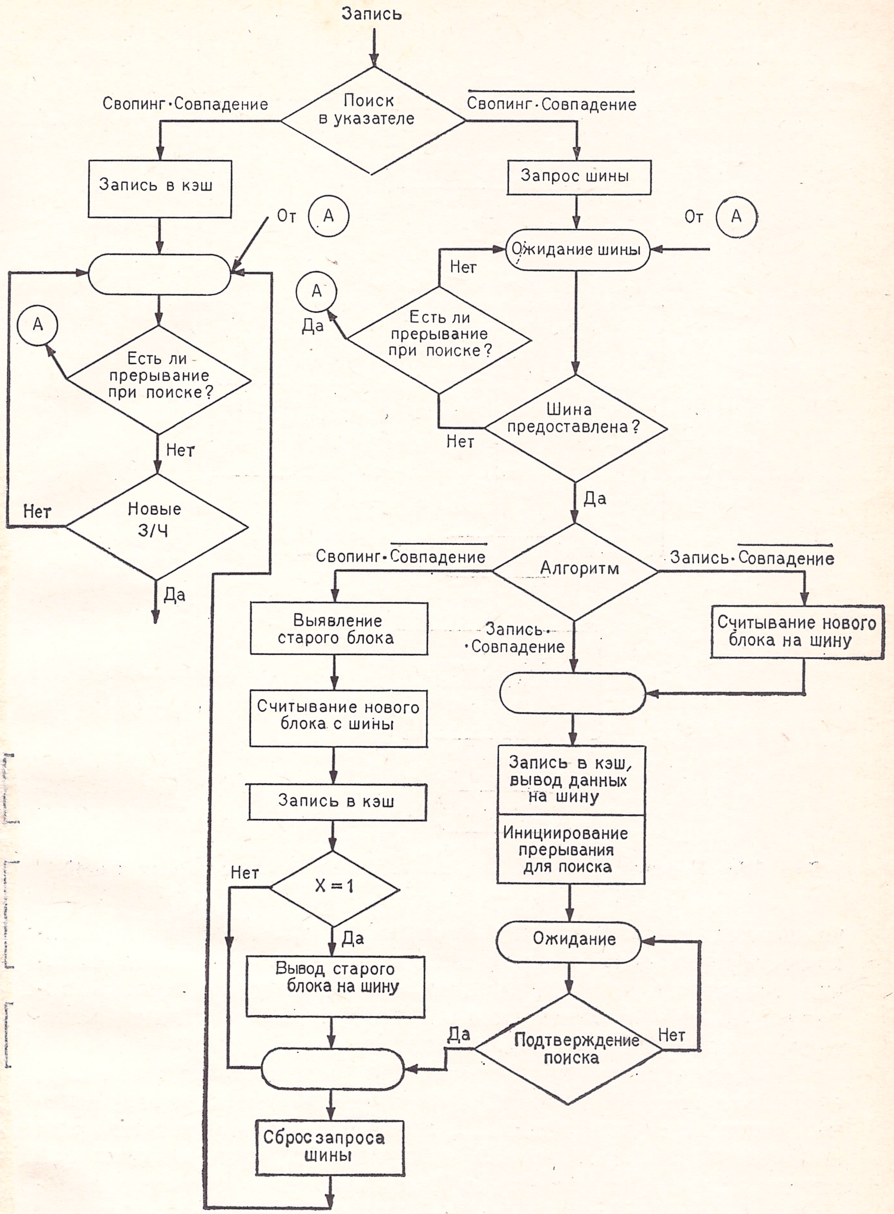


Рис. 5.8б. Управление записью в кэш.

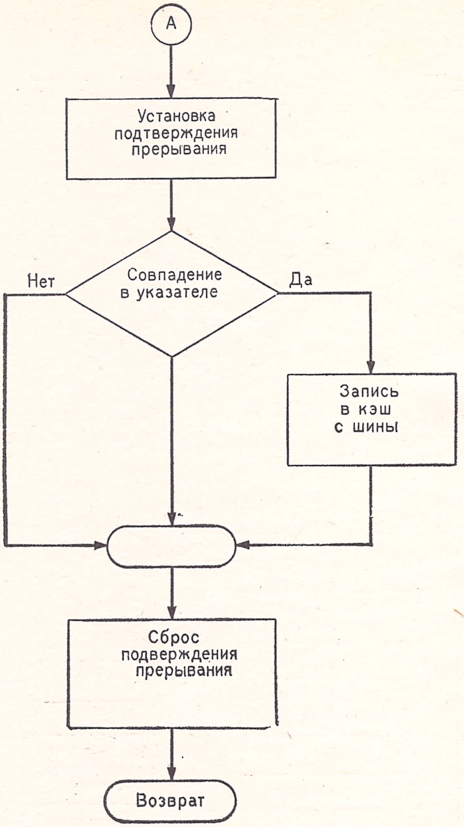


Рис. 5.8в. Управление многократным изменением содержимого кэшей.

уведомления в моменты времени, когда процессоры запрашивают перераспределения. В любом из этих вариантов наличие соответствующих полей защиты позволяет сделать аппаратную организацию главной памяти не зависящей непосредственно от числа процессоров.

Эффективность данного подхода можно легко оценить, сравнивая шинный трафик, инициируемый процессором, который использует этот подход, с трафиком, имеющим место при использовании процессором обычного алгоритма сквозной записи. Из соображений удобства анализа обращения к локальной памяти учитывать не будем, так как независимо от применяемого алгоритма свопинга эти обращения не сказываются на интенсивности загрузки шины.

Рассмотрим следующие «усредненные» параметры, характеризующие процессоры с кэш-памятью:

M_r — вероятность неудачного обращения к кэшу;
f_r — доля операций чтения в общем числе обращений;
x — доля блоков, находящихся в кэше и подвергающихся изменениям;
S_f — доля обращений к распределенной памяти, затрагивающих блоки главной памяти;
B_T — интенсивность шинного трафика.

Если для работы с кэшем используется простой алгоритм свопинга со сквозной записью, то шинный трафик, инициируемый процессором (B_{TW}) и выражаемый через интенсивность его обращений к кэш-памяти (R_m), описывается выражением

$$B_{TW} = \{[M_r(f_r + (1 - f_r)\,2) + (1 - M_r)(1 - f_r)]\}\,R_m = \\ = (1 - f_r + M_r)\,R_m.$$

Данная формула подразумевает, что обычно запись затрагивает не более одного блока и нужный блок перемещается в буфер при необходимости выполнения записи, если к моменту записи его там не было (возможен и вариант, когда такого перемещения не производится). Для рассматриваемого случая алгоритма с переключением величина трафика с учетом доли обращений к распределенной памяти (S_f) определяется следующим образом:

$$B_{TS} = S_f(1 - f_r + M_r)\,R_m + (1 - S_f)\,M_r(1 + x)\,R_m.$$

Если не принимать во внимание управляющий трафик и полностью локальные обращения, то два приведенных выражения описывают шинный трафик, инициируемый процессором, на основе определенных выше параметров. Изменение коэффициента уменьшения трафика (B_{TW}/B_{TS}) в зависимости от доли числа обращений к распределенной памяти (S_f) при номинальных значениях параметров иллюстрируется на рис. 5.9. Важно отметить, что небольшие значения доли обращений к распределенной памяти делают возможным относительное уменьшение шинного трафика на порядок и даже более. При этом можно на порядок увеличить количество процессоров в системе до того, как наступит «насыщение» шины.

Рис. 5.9 позволяет заметить, что, если уменьшать вероятность неудачных обращений за счет увеличения объема буфера, то могут быть достигнуты и другие преимущества. При вероятности неудачных обращений 0,01 и доле обращений к распределенной памяти 0,04 шинный трафик при использовании рассматриваемого метода снижается в 50 раз по сравнению с ситуацией, когда процессор не снабжен кэшем.

Если бы запросы кэшей на шину подчинялись пуассоновскому распределению и количество подключенных к шине про-

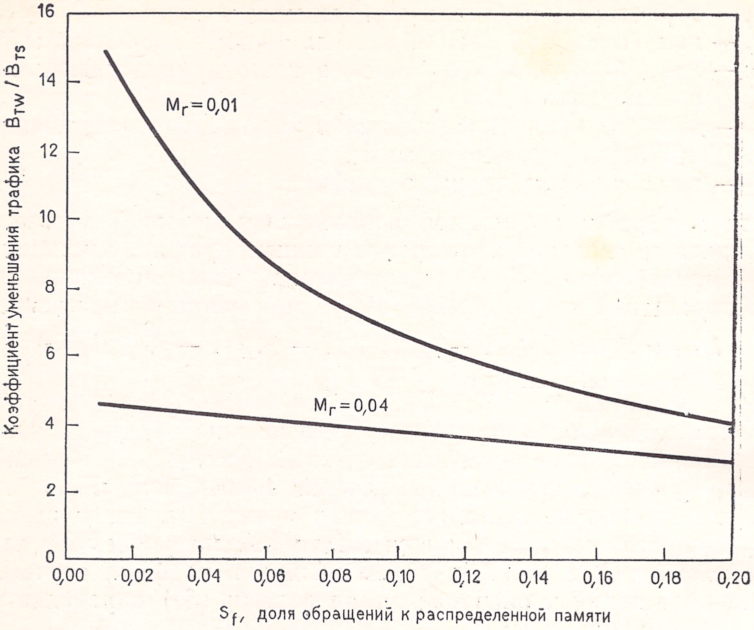


Рис. 5.9. Зависимость коэффициента уменьшения трафика от числа обращений к распределенной памяти.

B_{TW} — шинный трафик при использовании алгоритма сквозной записи; B_{TS} — шинный трафик при использовании алгоритма с переключением; $f_r = 0{,}8$; $x = 0{,}2$.

цессоров было таково, что определяемый ими при их полной одновременной занятости трафик совпадал с максимальными возможностями шины, то производительность системы составляла бы в зависимости от числа процессоров от 80 до 90 % максимально возможной. При введении в систему нескольких новых процессоров можно приблизиться к использованию предельных возможностей шины за счет увеличения времени ожидания процессорами доступа к шине.

Так, например, если бы 20 полностью занятых процессоров инициировали трафик, соответствующий пропускной способности шины, то шина и каждый из процессоров были бы заняты 81 % времени. Если число процессоров увеличить до 25, то шина будет занята 93,5 % времени, а каждый процессор — 75 % времени. Проверить справедливость этих данных можно с помощью простой модели очереди, рассматриваемой в следующем разделе.

В анализируемой системе процессор, в распоряжении которого находится шина, прерывает обращения всех других процессоров к кэшам только в случае, когда имеет место запись в общие блоки. Поэтому большинство шинных пере-

сылок не приводит к прерываниям. Благодаря тому что кэш-память имеет более короткий цикл обращения, чем главная память, поиск в указателях и обновление содержимого памяти могут происходить одновременно с шинными пересылками с записью, при которых в работу вовлекаются распределенные блоки памяти. То обстоятельство, что доступ к шине в любой момент времени предоставляется только одному процессору, обусловливает возможность обновления совместно используемой информации в любой момент времени лишь этим единственным процессором.

Методы использования кэш-памяти с помощью переключательного алгоритма характеризуются рядом преимуществ. Так, достаточно велико ожидаемое снижение шинного трафика, инициируемого процессорами. Процессоры могут быть введены в систему или удалены из нее без внесения изменений в ее аппаратные средства. Предполагается лишь добавление небольших аппаратных средств в кэши и главную память. Кроме того, объем информации, необходимой для управления шиной, меньше, чем при использовании более сложных методов [С3]. Доля обращений к памяти, при которых имеет место операция записи, лежит, как правило, в пределах 10—30 %. Можно предполагать, что доля обращений к распределенной памяти окажется небольшой. Области памяти, которые включают распределенную память, обычно представляют собой коммуникационные буферы, общие файлы и другие подобные средства хранения информации, которые с большой степенью вероятности допускают применение множества активных буферных копий. Использование более сложных схем организации памяти, в которых применяются дубликаты указателей или флаги присутствия, позволило бы уменьшить шинный трафик лишь в тех случаях, когда общий блок находится при операции записи в одном кэше. В большинстве реальных применений это имеет место нечасто.

Рассмотренная схема, в которой подчеркивается роль шины и главной памяти как факторов, ограничивающих повышение производительности, позволяет уменьшить интенсивность шинного трафика ценой небольших потерь в использовании возможностей процессоров, что отличает ее от схем, организованных в соответствии с методами Тэнга, Сенсье и Фотрие.

5.5. ПРОПУСКНАЯ СПОСОБНОСТЬ ШИН ПРИ ОБСЛУЖИВАНИИ МУЛЬТИПРОЦЕССОРОВ

Пропускная способность шины при обслуживании ею процессоров может быть проанализирована на основе представления процессоров и шины в виде объектов и средств обслу-

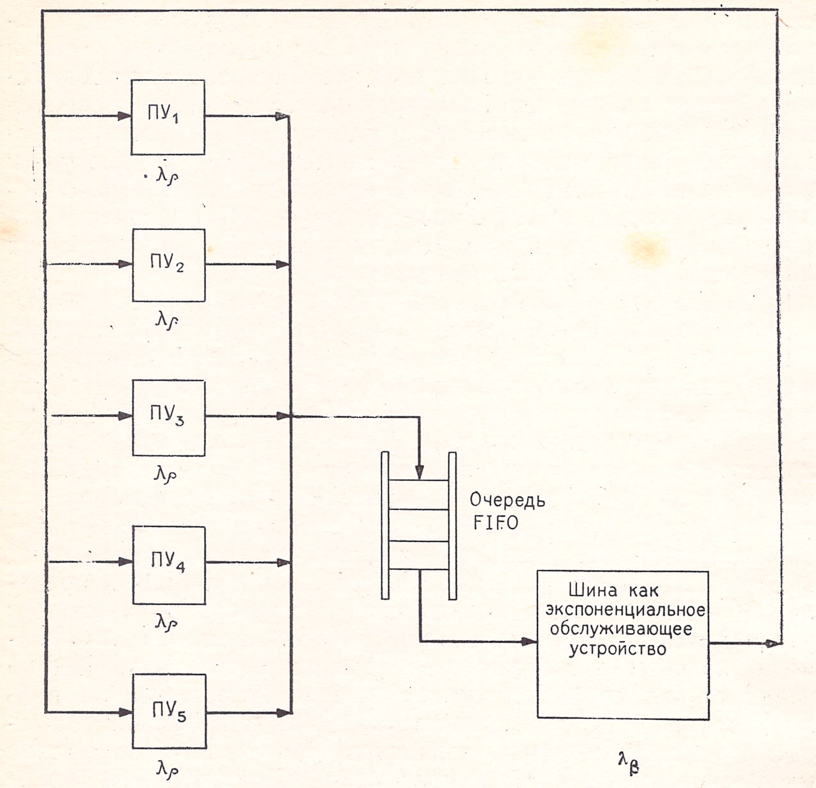


Рис. 5.10. Организация очереди обработки запросов процессоров на использование шины.

λ_ρ — интенсивность запросов со стороны процессора; λ_β — интенсивность обслуживания шиной процессоров.

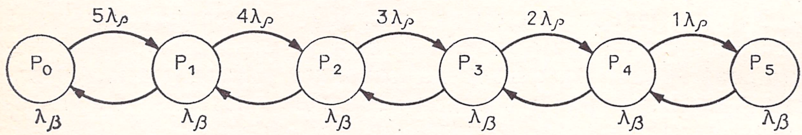


Рис. 5.11. Диаграмма обслуживания очереди процессоров.

Вероятности: P_0 — запросов шины нет, P_1 — один запрос шины, P_2 — два запроса шины, P_3 — три запроса шины, P_4 — четыре запроса шины, P_5 — пять запросов шины.

живания с экспоненциальными характеристиками. На рис. 5.10 показаны пять процессорных устройств (ПУ), каждое из которых, пребывая в рабочем состоянии, характеризуется интенсивностью обращений к шине λ_ρ. Предполагается, что шина имеет скорость обслуживания λ_β. Схема построения

очередей в такой системе приведена на рис. 5.11. Обратите внимание, что если все процессоры заняты, то общая скорость выработки запросов в пять раз превышает интенсивность их инициирования одним процессором. Далее, в этом простом примере подразумевается, что каждый процессор охватывает расчлененное пространство памяти, и поэтому глобальный поиск в указателе производить не требуется.

Если пренебречь частями решения, соответствующими переходному процессу, то условия равновесия во времени в среднем для вероятностей P_0, P_1, P_2, P_3, P_4 и P_5 могут быть записаны следующим образом:

$$P_0+P_1+P_2+P_3+P_4+P_5=1,$$

$$\frac{dP_0}{dt}=-5\lambda_\rho P_0+P_1\lambda_\beta=0,$$

$$\frac{dP_1}{dt}=-P_1\lambda_\beta-4\lambda_\rho P_1+5\lambda_\rho P_0+P_2\lambda_\beta=0,$$

$$\frac{dP_2}{dt}=-P_2\lambda_\beta-3\lambda_\rho P_2+4\lambda_\rho P_1+P_3\lambda_\beta=0,$$

$$\frac{dP_3}{dt}=-P_3\lambda_\beta-2\lambda_\rho P_3+3\lambda_\rho P_2+P_4\lambda_\beta=0,$$

$$\frac{dP_4}{dt}=-P_4\lambda_\beta-\lambda_\rho P_4+2\lambda_\rho P_3+P_5\lambda_\beta=0,$$

$$\frac{dP_5}{dt}=-P_5\lambda_\beta+P_4\lambda_\rho=0.$$

Решение этих уравнений относительно вероятностей дает

$$P_0=\frac{1}{\frac{1}{1}+\frac{5\lambda_\rho}{\lambda_\beta}+\frac{5\times4\lambda_\rho^2}{\lambda_\beta^2}+\frac{5\times4\times3\lambda_\rho^3}{\lambda_\beta^3}+\frac{5\times4\times3\times2\lambda_\rho^4}{\lambda_\beta^4}+\frac{5\times4\times3\times2\times\lambda_\rho^5}{\lambda_\beta^5}}=\frac{1}{D},$$

$$P_1=\frac{5\lambda_\rho/\lambda_\beta}{D},\qquad P_2=\frac{5\times4\lambda_\rho^2/\lambda_\beta^2}{D},$$

$$P_3=\frac{(5\times4\times3\lambda_\rho^3)/\lambda_\beta^3}{D},\qquad P_4=\frac{(5\times4\times3\times2\lambda_\rho^4)/\lambda_\beta^4}{D},$$

$$P_5=\frac{(5\times4\times3\times2\lambda_\rho^5)/\lambda_\beta^5}{D}.$$

Если, например, скорость обслуживания шиной процессоров λ_β была равна $5\lambda_\rho$, то шина простаивала бы (P_0) лишь 28,5 % времени, а каждый из процессоров был занят 71,5 %

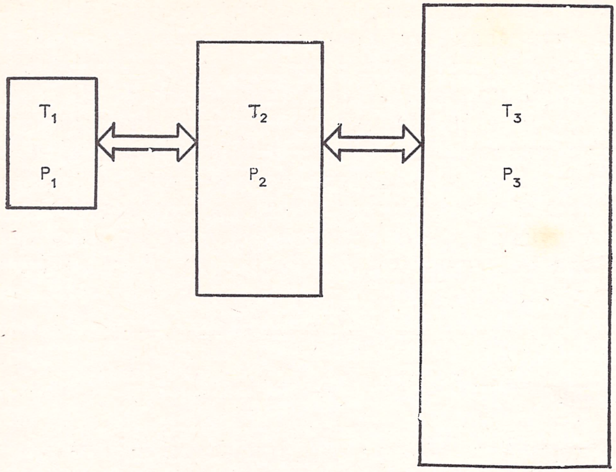


Рис. 5.12. Трехуровневая иерархия памяти.
T_i — время цикла, P_i — цена одного слова, $T_1 < T_2 < T_3$; $P_1 > P_2 > P_3$

времени. При $\lambda_\beta = 4\lambda_\rho$ шина простаивала бы 19,9 % времени, а каждый процессор был бы занят 64,1 % времени.

Как видно из выражения для времени простоя шины (P_0), для уменьшения этого времени можно увеличивать число процессоров в системе или применять более быстродействующие процессоры; однако при этом увеличивается время простоя каждого процессора.

5.6. ОПТИМИЗАЦИЯ ТРЕХУРОВНЕВЫХ ИЕРАРХИЧЕСКИХ СТРУКТУР

До сих пор рассматривались только двухуровневые иерархические структуры. Однако соответствующий уровень развития технологии позволяет создавать оптимальные системы, содержащие три или бо́льшее число уровней. Ниже проводится анализ оптимального распределения ресурсов в таких системах.

На рис. 5.12 иллюстрируется построение трехуровневой иерархической структуры, в которой для взаимодействия двух первых уровней используется алгоритм свопинга с флагами. Предполагается, что для получения новых блоков в данной иерархической системе применяется метод простых запросов. Приняты также следующие определения и предположения:

T_i — время цикла на i-м уровне;
P_i — стоимость одного слова на i-м уровне;
W_i — число слов на i-м уровне памяти;

D_i — затраты в долларах, приходящиеся на i-й уровень;

S — фиксированные суммарные затраты в долларах, приходящиеся на первые два уровня;

$M_{ri} = \frac{a^\alpha}{(a + D_i/P_i)^\alpha}$ — принятая аппроксимация для вероятности неудачных обращений как функции D_i/P_i слов данного уровня;

x — доля блоков, отмеченных флагами, полагаемая одинаковой на первых двух уровнях.

Оптимизация системы будет заключаться в распределении фиксированных суммарных затрат в долларах с таким расчетом, чтобы минимизировать эффективное время цикла ($T_{эфф}$). Выражение для эффективного времени цикла, приводимое ниже, получено в предположении, что в памяти более высокого уровня содержатся все слова, имеющиеся в запоминающих устройствах более низких уровней.

$$T_{эфф} = (1 - M_{r1}) T_1 + (H_{r2} - H_{r1}) [T_1 + (1 + x) T_2] + \\ + (1 - H_{r2}) [T_1 + T_2 + (1 + x) T_3] = \\ = T_1 + M_{r1} (1 + x) T_2 + M_{r2} [T_3 (1 + x) - xT_2] = \\ = T_1 + M_{r1} (1 + x) T_2 + M_{r2} [(1 + x) T_3'], \text{ где}$$

$$T_3' = T_3 - \frac{x}{(1 + x)} T_2.$$

Имея в виду, что при фиксированных ресурсах

$$S = W_1 P_1 + W_2 P_2 = D_1 + D_2,$$

и подставляя в выражение для $T_{эфф}$ выражения M_{r1} и M_{r2}, получаем

$$T_{эфф} = T_1 + \frac{a^\alpha (1 + x) T_2}{\left(a + \frac{D_1}{P_1}\right)^\alpha} + \frac{a^\alpha (1 + x) T_3'}{\left(a + \frac{S - D_1}{P_2}\right)^\alpha}.$$

Дифференцируя $T_{эфф}$ по D_1 и приравнивая производную нулю для нахождения минимума, получаем

$$\frac{dT_{эфф}}{dD_1} = - \frac{\alpha a^\alpha (1 + x) T_2}{P_1 \left(a + \frac{D_1}{P_1}\right)^{\alpha + 1}} + \frac{\alpha a^\alpha (1 + x) T_3'}{P_2 \left(a + \frac{S + D_1}{P_2}\right)^{\alpha + 1}} = 0.$$

Решение этого уравнения относительно D_1 имеет вид

$$D_1 = \frac{S - a \left[\left(\frac{P_1}{P_2} \frac{T_3}{T_2}\right)^{1/(\alpha + 1)} - 1\right] P_2}{1 + \left[\left(\frac{P_2}{P_1}\right)^\alpha \frac{T_3}{T_2}\right]^{1/(\alpha + 1)}}.$$

В общем случае значение второго члена числителя в диапазоне значений, представляющем интерес, пренебрежимо мало

по сравнению с S. С учетом этого получаем следующее приближенное выражение для оптимального распределения ресурсов:

$$D_1 = \frac{S}{1 + [(P_2/P_1)^{\alpha} (T_3/T_2)]^{1/(\alpha+1)}}.$$

Стоимость памяти, как выясняется эмпирическим путем, увеличивается примерно прямо пропорционально квадратному корню быстродействия. Если эту зависимость использовать для установления стоимости памяти второго уровня иерархии, то получим $P_2 = \sqrt{T_1/T_2} P_1$. С помощью этой зависимости можно также определить оптимальное время цикла памяти второго уровня. Подстановка P_2 в выражение для $T_{эфф}$ дает

$$T_{эфф} = T_1 + \frac{a^{\alpha}(1+x)\,T_2}{\left(a + \dfrac{D_1}{P_1}\right)^{\alpha}} + \frac{a^{\alpha}(1+x)\,T_3'}{\left(a + \dfrac{(S-D_1)}{P_1}\dfrac{T_2^{1/2}}{T_1^{1/2}}\right)^{\alpha}}.$$

Минимальное значение этого выражения можно найти, приравнивая нулю его частные производные по D_1 и T_2. Дифференцирование по D_1 приводит к полученному выше результату. Дифференцирование по T_2 дает

$$\frac{dT_{эфф}}{dT_2} = \frac{a^{\alpha}(1+x)}{\left(a + \dfrac{D_1}{P_1}\right)^{\alpha}} - \frac{\alpha a^{\alpha} T_3' (1+x)(S-D_1)}{2T_1^{1/2}T_2^{1/2}P_1\left[a + \dfrac{S-D_1}{P_1}\dfrac{T_2^{1/2}}{T_1^{1/2}}\right]^{\alpha+1}} = 0.$$

Полагая, что $a \ll D_1/P_1$ или $(S - D_1)/P_1$, и преобразуя это выражение, находим

$$T_2^{(\alpha+2)/2} = \frac{\alpha}{2}\left(\frac{D_1}{S-D_1}\right)\alpha T_1^{\alpha/2} T_3'.$$

Принимая во внимание, что

$$\frac{D_1}{S-D_1} = \frac{T_2(2+\alpha)/2(\alpha+1)}{T^{\alpha/(\alpha+1)}T^{1/(\alpha+1)}},$$

получаем

$$T_2 = \left(\frac{\alpha}{2}\right)^{[2(\alpha+1)/(\alpha+2)]} T_1^{1/(\alpha+2)} T_3'^{[2/(\alpha+2)]}.$$

В качестве более конкретного примера рассмотрим супер-ЭВМ, у которой память первого уровня имеет время цикла 10 нс, быстродействие памяти второго уровня может варьироваться, а время цикла памяти третьего уровня составляет 2560 нс. Полагая $\alpha = 1$, определим оптимальное значение T_2

$$T_2 = (1/2)^{4/3} \times 10^{1/3} \times 2560^{2/3} = 160 \text{ нс}.$$

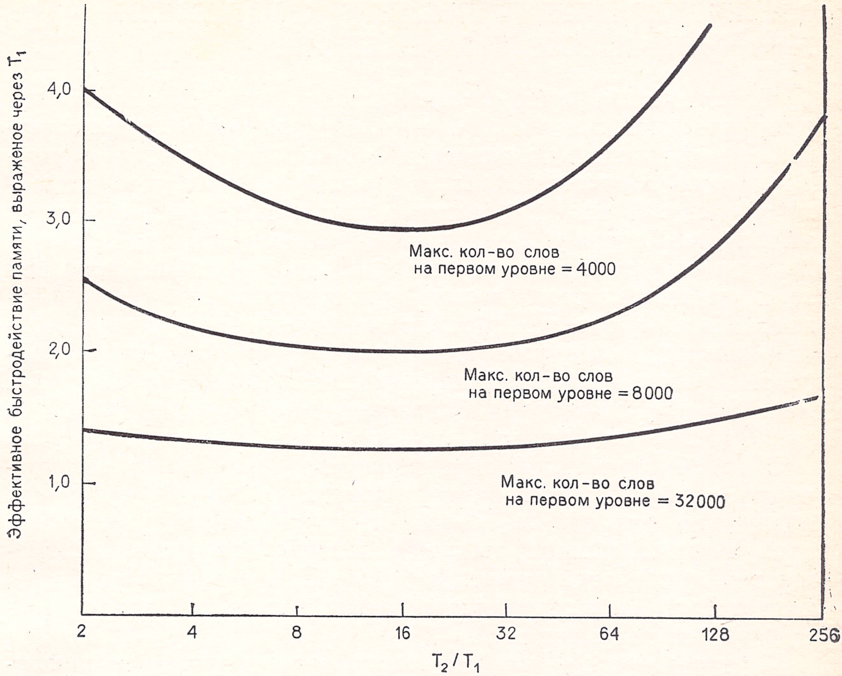


Рис. 5.13. Зависимость эффективного быстродействия памяти от соотношения быстродействий первого и второго уровней (T_2/T_1).

$$T_3 = 256\,T_1, \quad P_2 = P_1\sqrt{T_1/T_2},$$
$$Mr_i = 50/(50 + W_i), \quad \alpha = 1.$$

На рис. 5.13 показана зависимость эффективного времени цикла данной системы (в качестве единицы измерения используется величина T_1) от S и T_2. На графиках отчетливо видна широкая область близких к минимуму значений эффективного времени цикла, располагающаяся в диапазоне значений T_2, близких к 160 нс.

На практике быстродействие и стоимость памяти описываются не столь плавно изменяющимися функциями. Часто кривые, представляющие взаимосвязь стоимости и производительности памяти, имеют резкие скачки. Однако наличие широкой области значений, близких к минимальному, на кривых, приведенных на рис. 5.13, свидетельствует о том, что быстродействие второго уровня не является критичным с точки зрения достижения высокой производительности системы памяти.

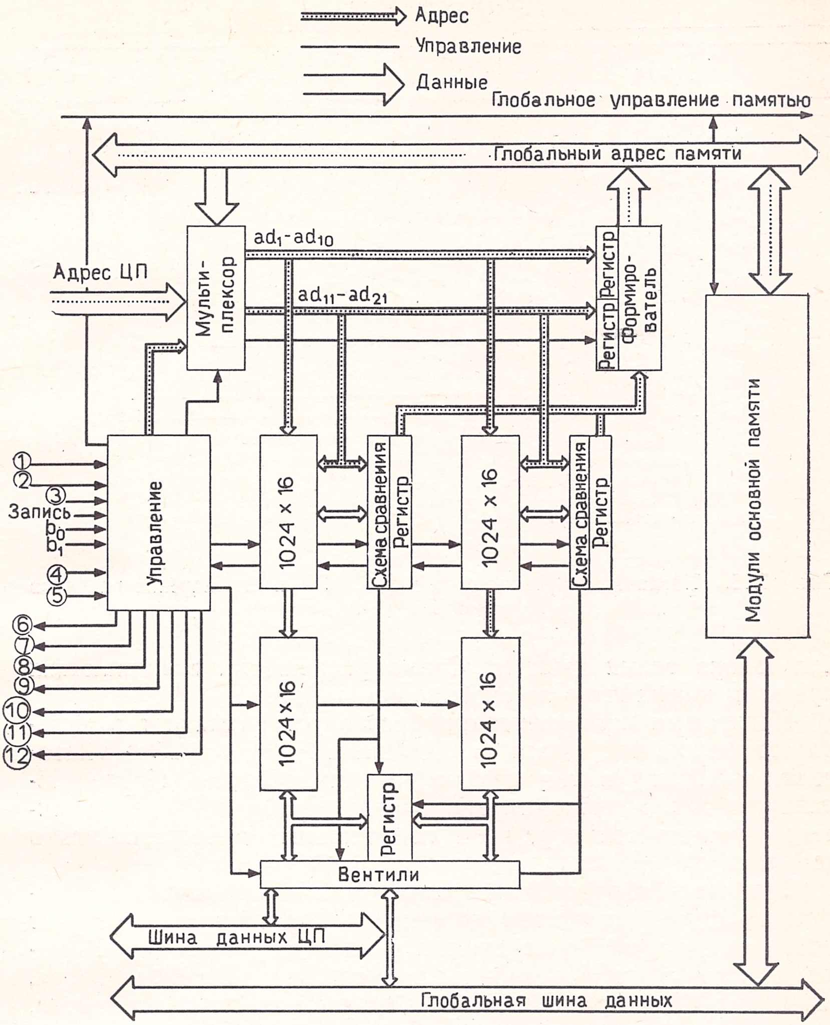
5.7. ПРИМЕР ПРОЕКТИРОВАНИЯ КЭШ-ПАМЯТИ МУЛЬТИПРОЦЕССОРНОЙ СИСТЕМЫ

Рассмотрим пример проектирования простой кэш-памяти для мультипроцессорной системы с одной шиной с использованием принципа многоклассовой организации, причем каждому классу сопоставлены два блока, содержащие по одному слову. В системе памяти используются LRU-стратегия замещения блоков и переключательный алгоритм замещения. Для работы с блоками памяти, не охваченными сквозной записью, применяется алгоритм регистрового свопинга с флагами. Из соображений простоты размер блока ограничен одним словом, а диапазон адресации — 22-битовой длиной адреса. Структурная схема памяти представлена на рис. 5.14. Рис. 5.15, *а* иллюстрирует назначение разрядов адреса, а рис. 5.15, *б* — назначение разрядов указателя.

В рассматриваемой системе памяти обеспечена возможность записи и слов, и байтов. Поэтому в случае неудачного обращения для записи, если затронут блок, для которого предусмотрен свопинг, текущее достоверное слово перемещается из основной памяти в буфер. Если речь идет о блоке, на который распространяется принцип сквозной записи, слово в кэш не перемещается.

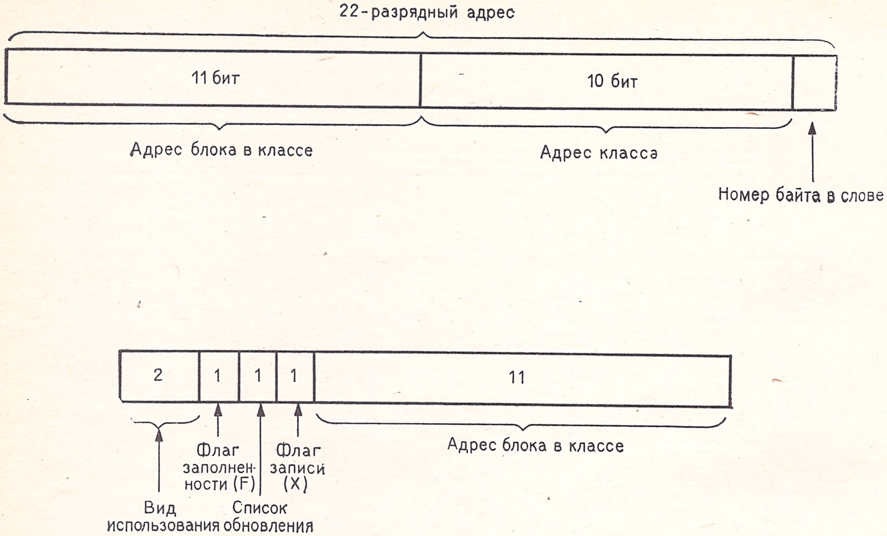
Предполагается, что каждому блоку сопутствует информация о виде использования. При попытке записи в отсутствующий блок эту информацию надо получить через шину из резидентной памяти до того, как могут быть продолжены действия с кэш-памятью. Поступающий из кэша сигнал «Статус блока неизвестен» информирует резидентную память через шину о необходимости получения битов, описывающих вид использования блока, и других необходимых сигналов. Например, последовательность использования шины при попытке записи в отсутствующий блок будет заключаться в следующих действиях. Сначала из главной памяти считывается нужный блок и в кэш вместе с управляющими сигналами посылаются биты состояния блока. Затем кэш, выяснив, что для блока предусмотрена сквозная запись, инициирует прерывание для глобального поиска и запись с использованием шины.

На рис. 5.16—5.18 представлены схемы алгоритмов управления чтением, записью и поиском информации за пределами кэша. Локальная синхронизация кэша осуществляется в данном случае синхросигналами процессора. Если длина шины составляет десятки футов, то, очевидно, шинные сигналы должны рассматриваться как асинхронные по отношению к локальным синхросигналам. Кроме того, необходимо принять меры для того, чтобы шина в равной степени могла быть ис-



① Разрешение на шину
② Просмотрено
③ Достоверность ЦП
④ Инициирование
⑤ Запрос на глобальный поиск
⑥ Достоверность шины
⑦ Шина занята
⑧ Запрос шины
⑨ Кэш готов
⑩ Достоверность данных кэша
⑪ Прерывание для поиска
⑫ Статус блока неизвестен

Рис. 5.14. Структура кэш-памяти,



б

Рис. 5.15. Назначение разрядов адреса (*а*) и назначение разрядов указателя (*б*).

пользована всеми кэшами. Схема разрешения возникающих при этом конфликтов на рисунке не приведена.

Программа инициализации просто устанавливает в нулевое состояние все флаги заполненности (F) в файле указателя, информируя тем самым, что кэши пусты. После начала работы происходит заполнение свободного пространства кэша до тех пор, пока не возникнет необходимость в свопинге.

Техническая реализация такой схемы предполагает обеспечение соответствующих временных характеристик за счет использования наиболее совершенных логических схем Шоттки и статических запоминающих устройств. Вообще говоря, имеется в виду применение серийно выпускаемых элементов, хотя и не все характеристики обязательно соответствуют данным конкретных изготовителей. Здесь приводятся лишь некоторые электрические характеристики, так как многие из них либо присущи только конкретным вариантам реализации, либо являются очевидными.

В рассматриваемую схему не включены средства проверки четности или исправления ошибок, которые обычно предусматриваются в составе выпускаемых изделий. На рис. 5.19 показан комплект ИС памяти и вентильных схем, необходимый для построения той части кэша, которая служит для размещения данных. В него входят ИС памяти с временем

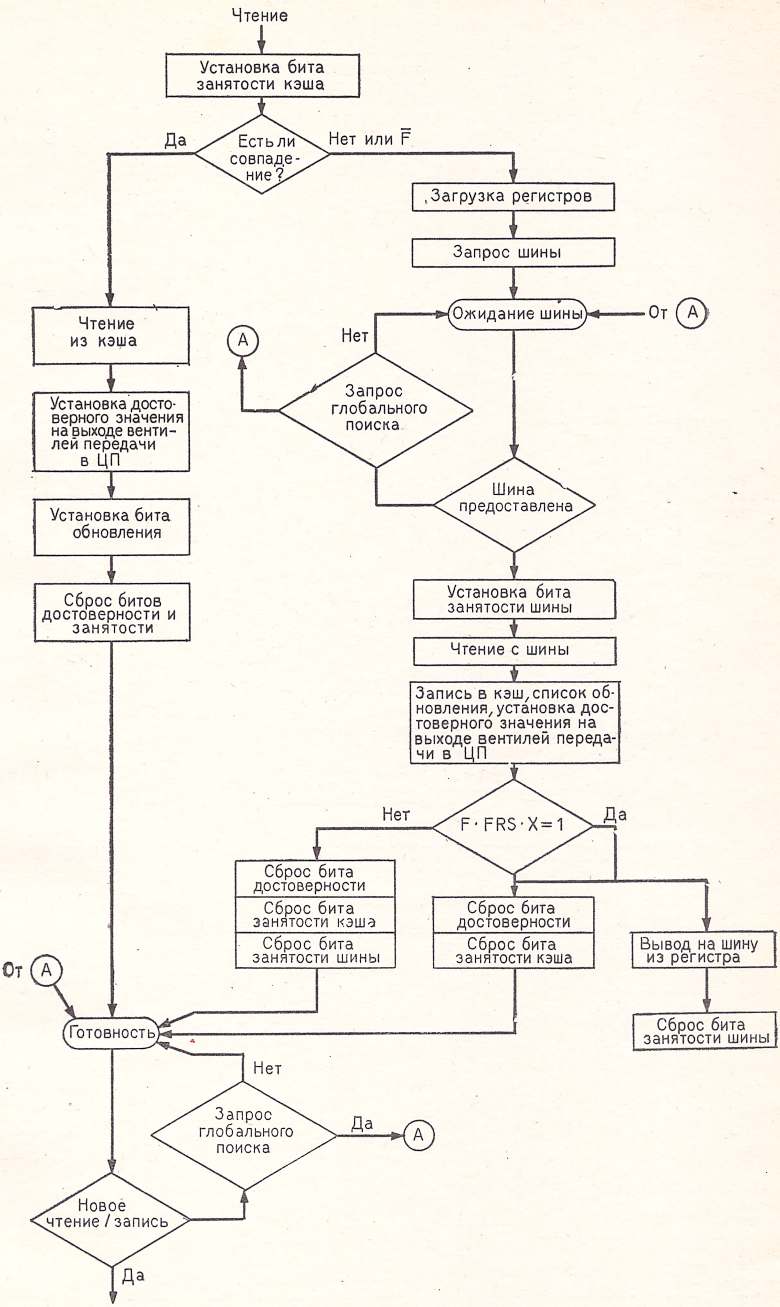


Рис. 5.16. Схема алгоритма управления чтением из кэш-памяти.

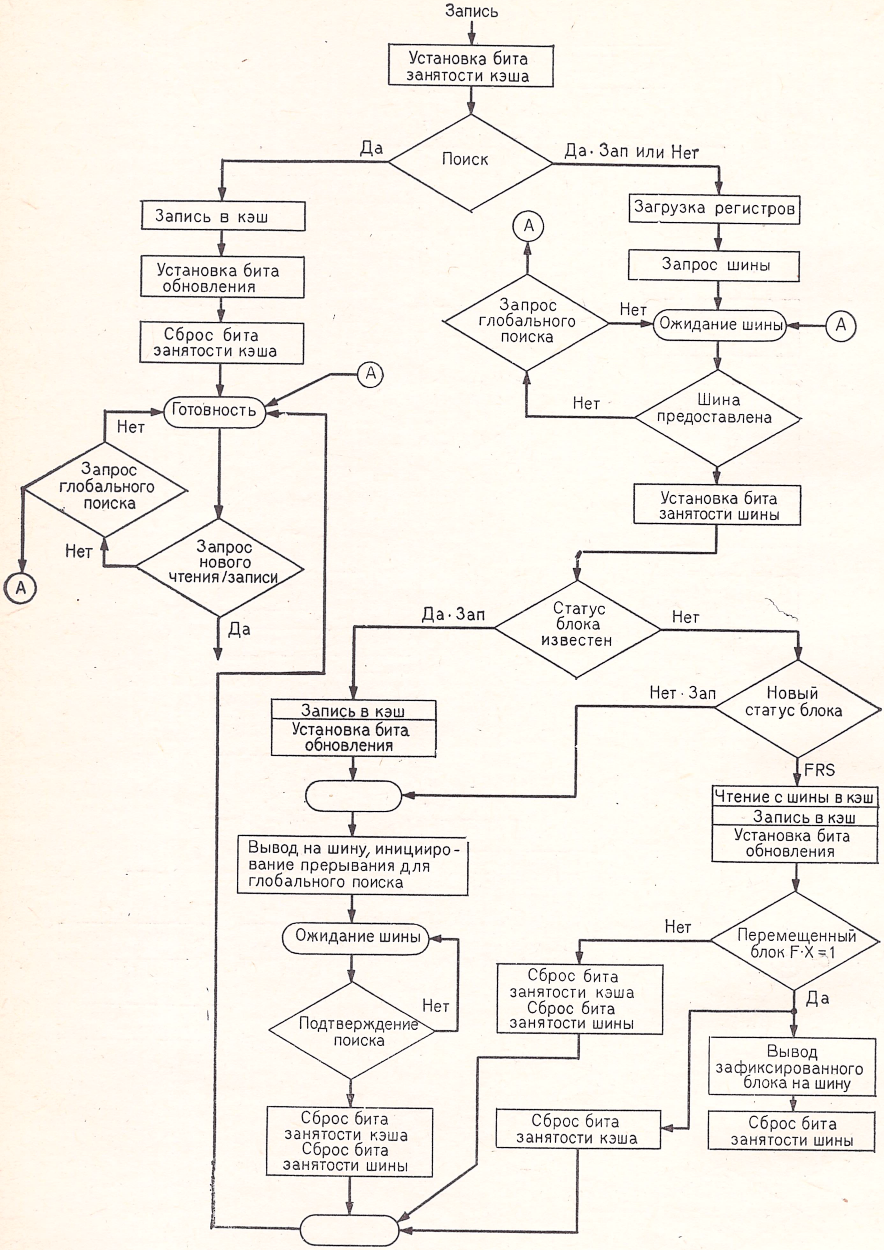


Рис. 5.17. Схема алгоритма управления записью в кэш-память.

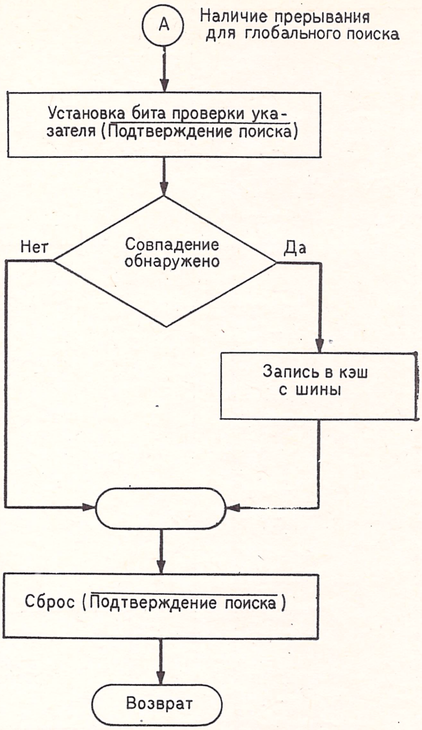


Рис. 5.18. Обслуживание прерывания для глобального поиска.

цикла 45 нс и временем выборки кристалла 20 нс, а также 16-разрядный регистр и вентили с тремя состояниями на выходе (см. рис. 5.19). Глобальная шина данных считается тристабильной. Схемы хранения данных, составляющие матрицу памяти, имеют мультиплексированные контакты ввода — вывода, в результате чего оказываются несколько сниженными как производительность, так и стоимость памяти.

На рис. 5.20 показан набор интегральных схем, необходимый для реализации указателя, списков обновления, флагов, регистров и схем сравнения. В связи с тем, что необходимо обеспечить выборочную запись в отдельные разряды слов, в данном случае применены ИС памяти 1К $\times$ 1 бит с временем цикла 25 нс. Эти ячейки памяти указателя, время обращения к которым меньше времени обращения к той части памяти, где хранятся данные, имеют быстродействие, достаточное для того, чтобы совместить во времени операции сравнения с указателем и операции обращения к данным. На рис. 5.21 показан набор ИС, обеспечивающий мультиплексирование и фиксацию адреса, а также управление шиной.

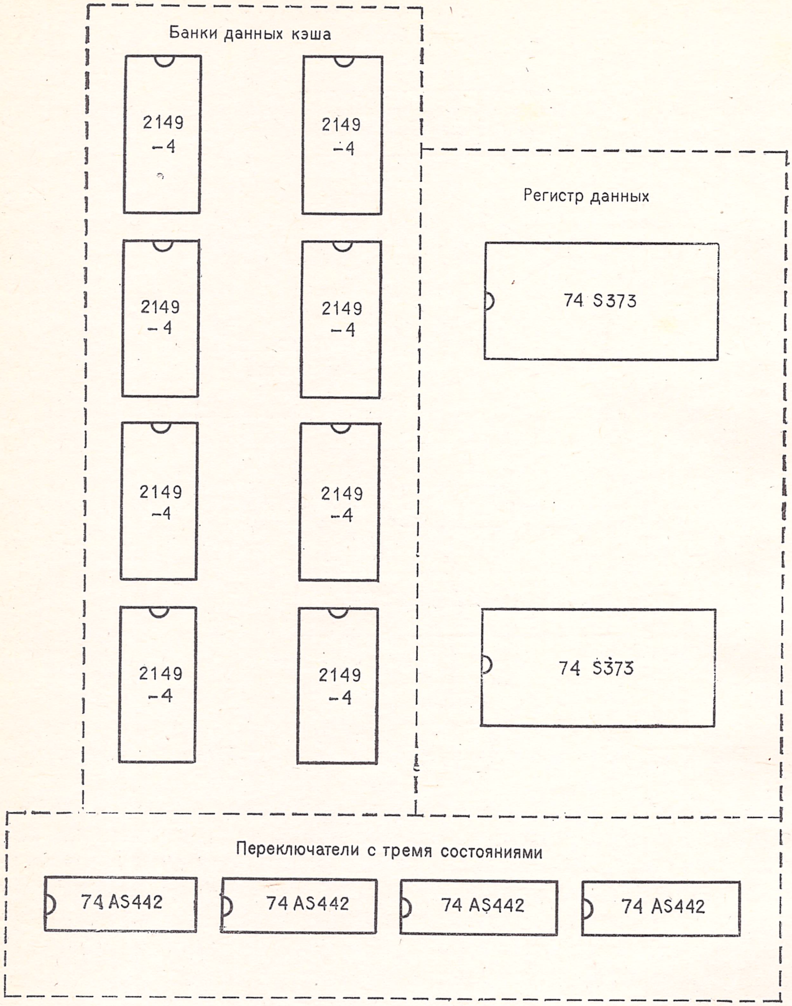


Рис. 5.19. Узлы кэша, служащие для хранения данных и управления потоками информации.

На рис. 5.22, *а* и *б* приведены временные диаграммы работы двух интегральных схем памяти, а на рис. 5.23 представлены соответствующие временные диаграммы при выполнении в рассматриваемой схеме операции чтения при условии удачного обращения. Аналогичные временные диаграммы можно было бы привести и для операции записи. При выполнении операции чтения в случае пребывания необходимых

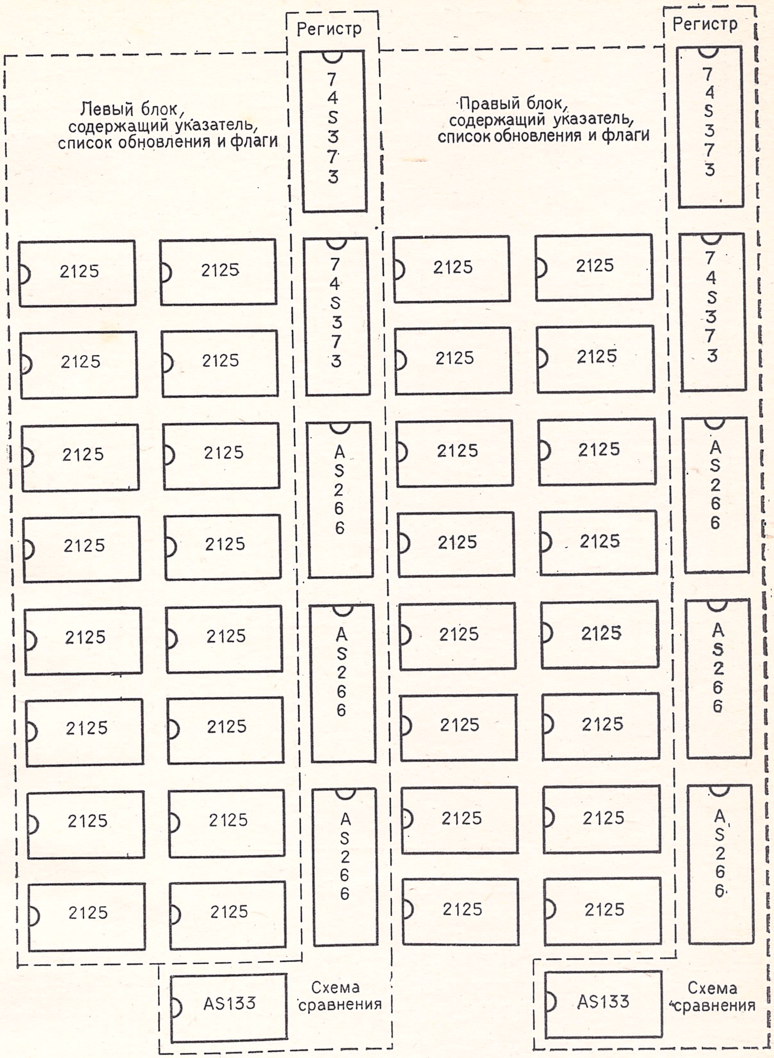


Рис. 5.20. Узлы кэша, служащие для размещения указателя, списка обновления и управляющих разрядов.

данных в кэше время обращения составляет для рассматриваемой схемы 67 нс, а время цикла 100 нс.

На рис. 5.24 управляющая логика представлена подробнее. В ее состав входит блок управления последовательностью включения питания и инициализацией. Последовательность инициализации включает установку в нулевое состояние всех

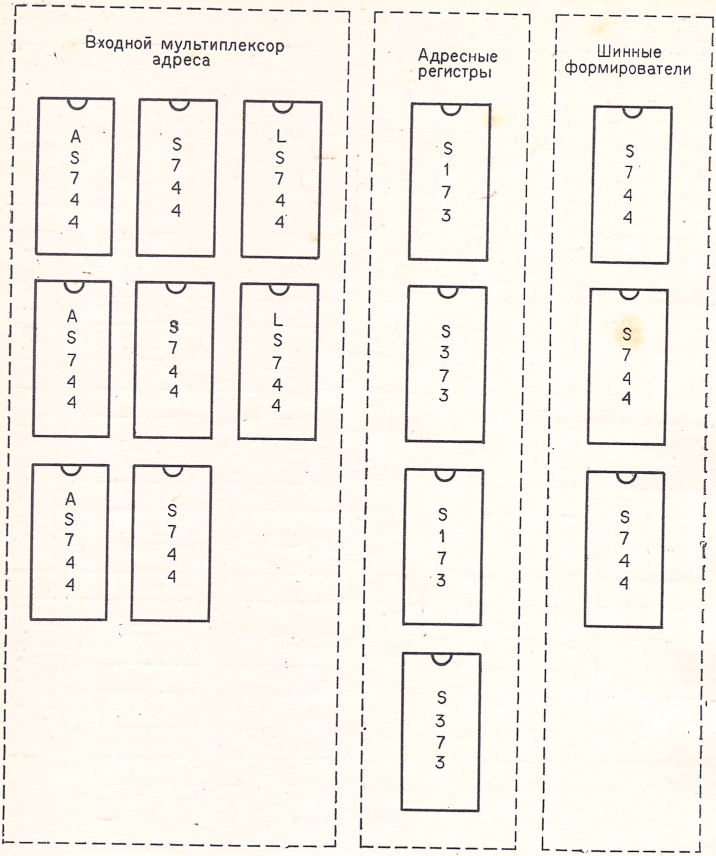
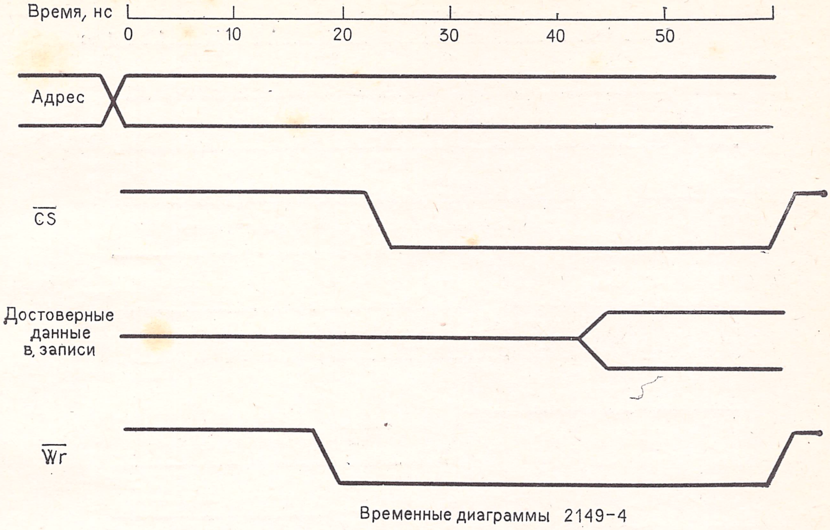


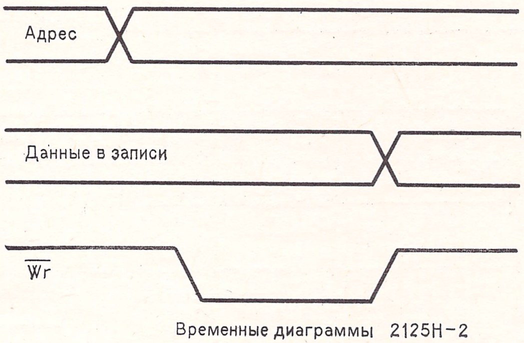
Рис. 5.21. Схема обработки адреса.

флагов заполненности и приведение системы в соответствующее состояние по сигналу запуска. Логика чтения, записи и прерывания для поиска служит для выработки управляющих сигналов системы и обеспечения временных соотношений при реализации алгоритмов управления, приведенных на рис. 5.16—5.18. Логика управления состоянием и потоками информации обеспечивает выработку сигналов, необходимых для управления регистрами, мультиплексорами, вентилями и т. д. Эту логику легко реализовать в виде совокупности триггеров, каждый из которых непосредственно соответствует одному из состояний, представленных в схеме алгоритма.

Этот рассмотренный не до конца пример построения памяти позволяет отметить, что при современном состоянии технологии количество интегральных схем, необходимое для



а



б

Рис. 5.22. Временные диаграммы работы кэш-памяти.

реализации кэш-памяти, невелико. Большинство используемых элементов составляют быстродействующие интегральные схемы памяти. Пример позволяет также заключить, что достаточно высокая производительность может быть достигнута во многих случаях на основе использования стандартных элементов.

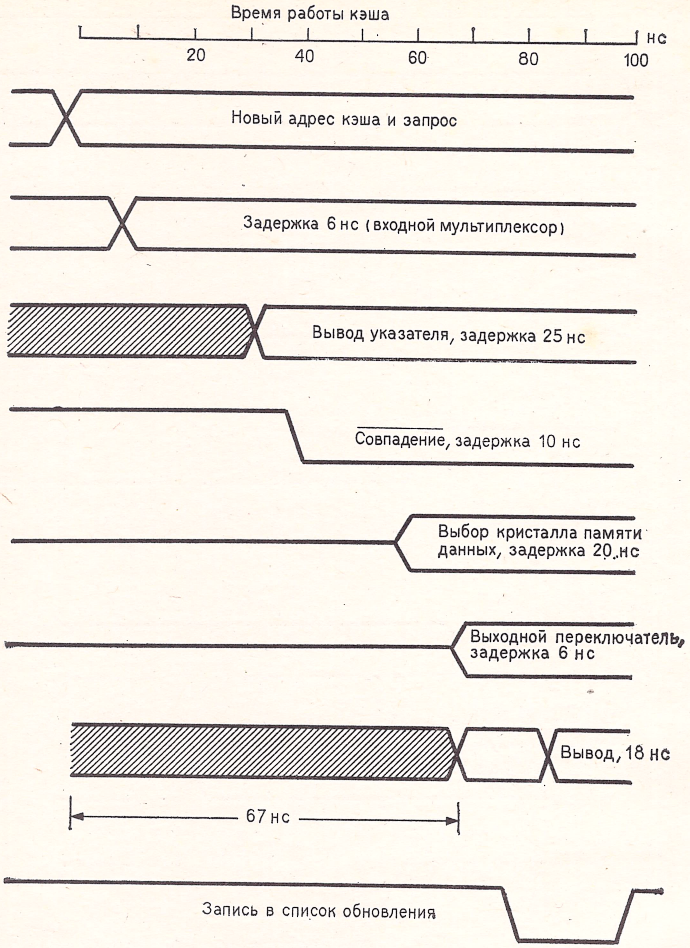


Рис. 5.23. Временные диаграммы операции чтения с использованием кэш-памяти при наличии совпадения с содержимым указателя.

5.8. СИСТЕМЫ ВИРТУАЛЬНОЙ ПАМЯТИ

5.8.1. Организация памяти и отображение информации

В больших мультипрограммных мультипроцессорных системах с архитектурой, подобной той, которая показана на рис. 5.25, внешняя память может иметь емкость 2^{32} байт ($N_S = 32$), диапазон адресации процессора может составлять 2^{24} байт ($Npa = 24$), а каждый процессор может при

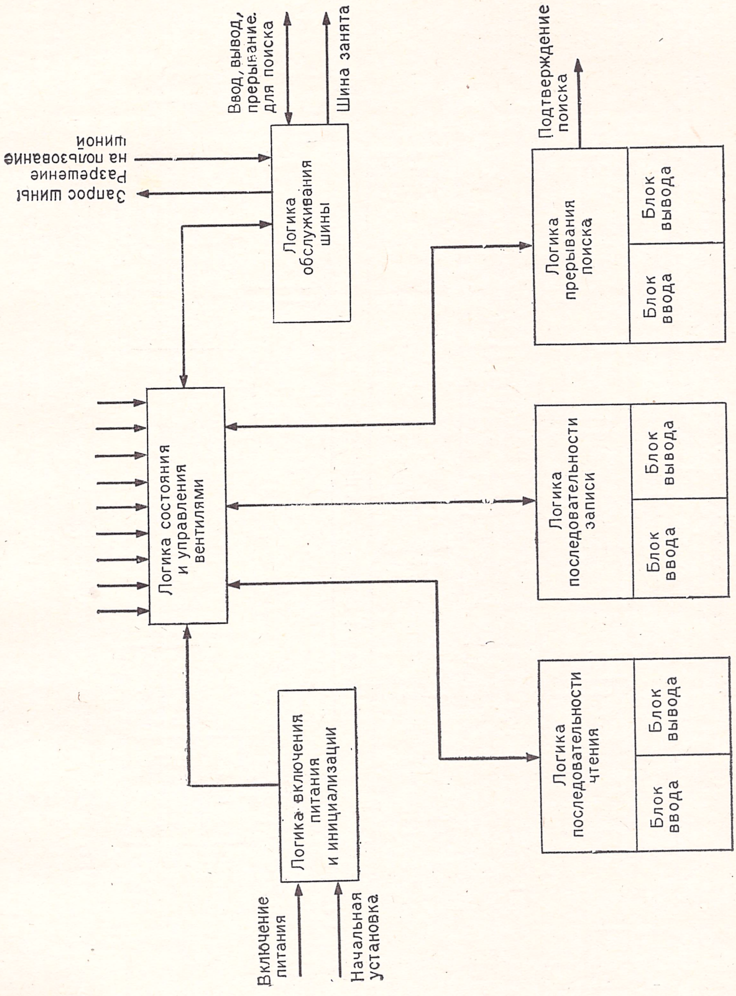


Рис. 5.24. Структурная схема устройства управления кэш-памяти.

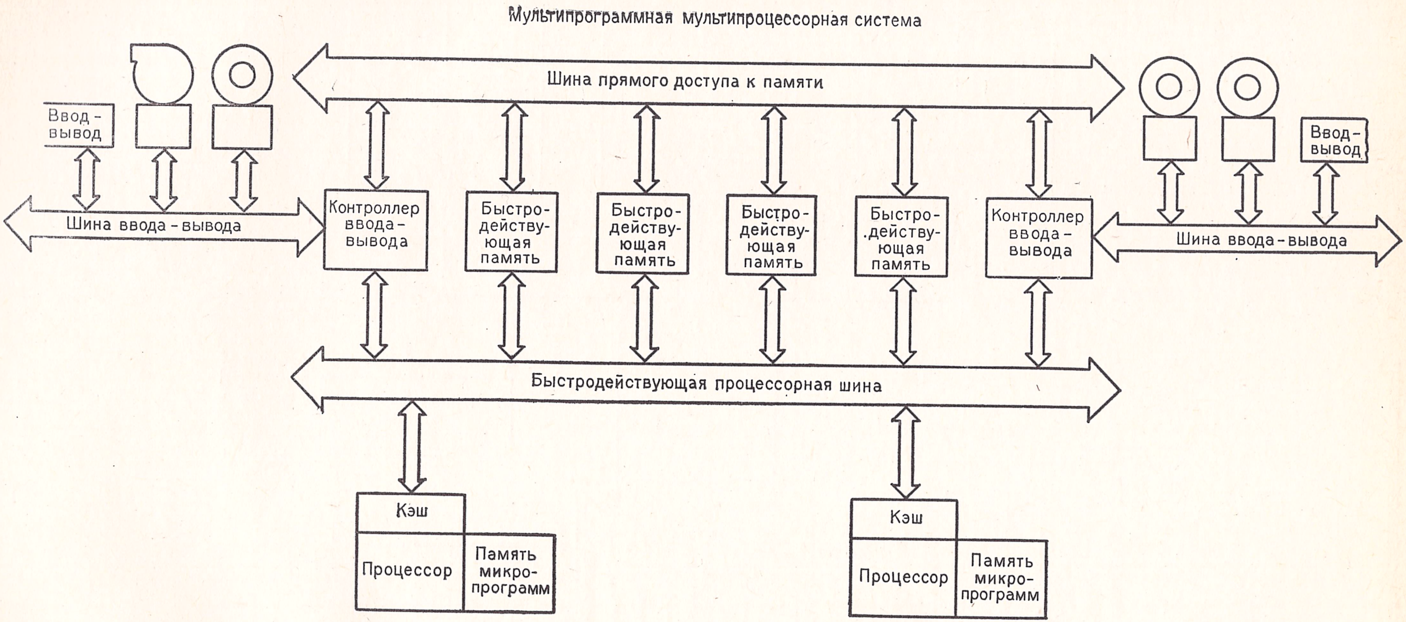


Рис. 5.25. Большая мультипрограммная мультипроцессорная система.

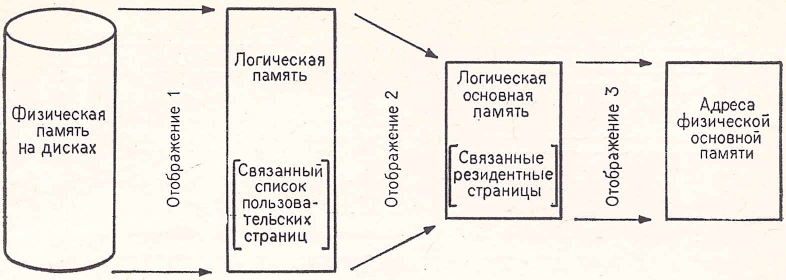


Рис. 5.26. Отображение информации в память.

этом занимать 2^{20} байт ($Np = 20$) основной памяти. В процессе работы система виртуальной памяти должна снабжать каждый процессор операндами и командами для всего многообразия решаемых им задач с такой скоростью, чтобы процессор был постоянно занят; при этом предполагается, что действия по вводу—выводу не ограничивают производительность вычислительной системы. Память, предоставляемая каждому процессору, должна распределяться между различными выполняемыми им заданиями, и нахождение наиболее часто используемых в каждом задании слов в основной памяти должно быть организовано таким способом, чтобы программисту не надо было об этом беспокоиться. Кроме того, некоторые информационные блоки должны использоваться совместно несколькими пользователями, а другие необходимо обеспечить защитой, чтобы применять их мог только владеющий ими пользователь. Довольно сложная система такого рода должна быть реализована экономически эффективно. Как отмечено Мэтиком [M11], при функционировании системы виртуальной памяти и системы внешних запоминающих устройств имеют место три вида отображения информации (рис. 5.26). Внешняя память обычно разделена на страницы, имеющие объем от 512 до 4096 байт, с целью оптимального использования возможностей запоминающих устройств на магнитных дисках и барабанах. (В рассматриваемом примере емкость составляет 4096 байт.) Внешняя память, отображенная в логическую память, предоставляется пользователям постранично, в результате чего на первом уровне отображения отсутствует необходимость адресации с точностью до байта. Кроме того, так как основная память процессора обычно представляет собой реально лишь часть основной памяти, которая могла бы быть адресована, соответствующая потребность в большей виртуальной основной памяти обычно удовлетворяется за счет иерархической организации, в составе которой имеются относительно медленные и дешевые диски и барабаны, позволяющие создать у пользо-

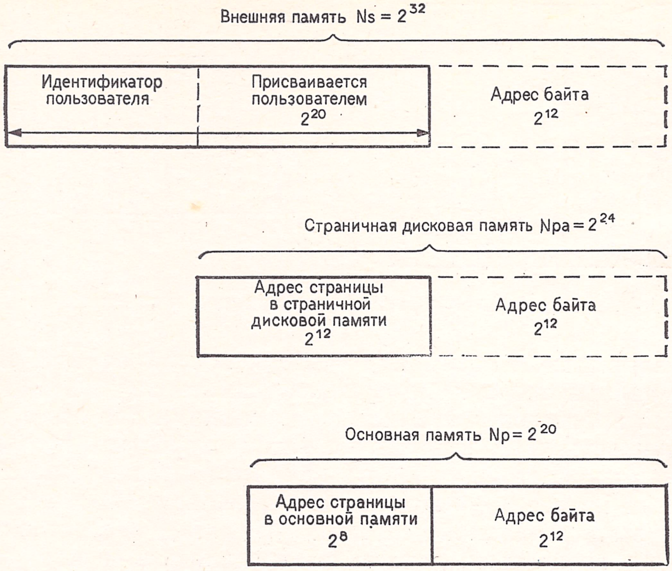


Рис. 5.27. Форматы адресов

вателя впечатление, что он работает с основной памятью большой емкости. Форматы адресов, используемых в описанной организации, показаны на рис. 5.27.

Программа одного пользователя, вообще говоря, может занять почти все адресуемое пространство памяти для размещения страниц задания, над которым пользователь работает. Однако, как правило, основной памятью одновременно совместно пользуются несколько заданий, пребывающих в активном состоянии.

С точки зрения общей организации основная память служит буфером для страничного диска, а страничный диск — рабочей памятью для обработки пользовательских задач, резидентных во внешней памяти. Таким образом, основной целью разработки памяти является такая ее организация, чтобы информация поступала в нее и из нее без серьезных препятствий. Страницы, отображенные из внешней логической пользовательской памяти в адресуемую логическую память процессора, составляют подмножество, непосредственно обрабатываемое в данный момент.

Вследствие относительно низкого быстродействия известных устройств внешней памяти бо́льшая часть операций, составляющих отображения 1 и 2 из числа показанных на рис. 5.26, может быть реализована программным путем. В то же время для выполнения многих операций, характерных для

отображения 3, необходимы аппаратные средства. Например, для получения информации, необходимой для связывания, могут быть использованы регистры базы и смещения; другие регистры, поставленные в соответствие блокам памяти, могут обеспечивать реализацию функций защиты и блокировки.

Ключевой проблемой разработки систем виртуальной памяти, в составе которых используются страничные диски, барабаны и другие аналогичные устройства, является преодоление большого расхождения между временем доступа к странице на страничном диске и временем обращения к быстродействующей основной памяти. Время обращения к основной памяти лежит обычно в диапазоне от 0,2 до 1 мкс, а эффективное время доступа к страничному диску составляет от 5 до 10 мс, т. е. имеет место отношение скоростей порядка 10^4. Применительно к многим задачам эффективный способ обеспечения постоянной занятости процессора заключается в применении мультипрограммирования. При мультипрограммной организации системы процессор, если для этого существуют условия, может начать обработку другого задания, пока в памяти происходит выборка нужной страницы. В этом случае обращение и пересылка страницы совмещаются во времени с исполнением процессором другого задания. В мультипроцессорной мультипрограммной системе могут возникать еще более сложные проблемы, когда, например, два процессора претендуют на одни и те же ресурсы (имеют место так называемые состязания). В результате этого производительность двухпроцессорной системы обычно превышает производительность однопроцессорной не более чем в 1,75 раза.

В других условиях обработки заданий (например, в интерактивном режиме) быстродействие страничных дисков оказывается в лучшем случае едва достаточным для обеспечения приемлемой работы системы. Потребность в страничной памяти с меньшим временем обращения является стимулирующим фактором для разработки запоминающих устройств, таких, например, как устройства на магнитных доменах.

5.8.2. Вероятность неудачного обращения за страницей

Как и в случае систем с кэш-памятью, критичным параметром систем виртуальной памяти является доля запросов памяти, при выполнении которых нужная страница не обнаруживается в основной памяти. Этот параметр принято называть *вероятностью неудачного обращения за страницей*, хотя аналогия между ним и рассмотренной ранее вероят-

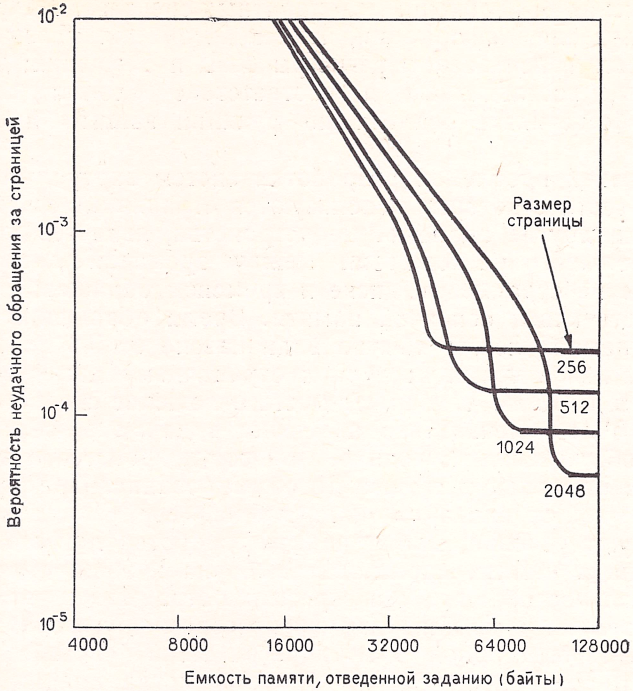


Рис. 5.28. Зависимость вероятности неудачного обращения за страницей от емкости отведенной памяти.
(Задача трансляции, стратегия LRU [C5].)

ностью неудачного обращения очевидна. Вероятность неудачного обращения за страницей, присущая работе с меньшими по размеру блоками, характерными для кэш-памяти, зависит от размера страницы; в основном же она определяется размерами сегментов памяти, отводимых для каждой задачи. На рис. 5.28 графически представлены типичные зависимости вероятности неудачного обращения от объема сегмента, отведенного заданию, которые получены Чу и Опдербеком [C5] и Мэттсоном [M10] для случая независимого выполнения одного задания. Экспериментальные результаты указывают на значительный разброс требований к памяти, предъявляемых со стороны задания. В общем случае вероятность неудачного обращения за страницей при увеличении числа страниц, отведенных заданию, резко убывает, пока не достигнет минимального значения. Это минимальное значение характеризует количество неудачных обращений, имеющих место в процессе перемещения задания, выполнения его и соответствующей записи результатов. Как видно на рис. 5.28, при

уменьшении размеров страниц минимальное значение вероятности неудачного обращения увеличивается, т. е. в процессе выполнения задания и разнесения результатов происходит бо́льшее число неудачных обращений. Если, однако, емкость основной памяти ограничена, более эффективное ее использование достигается при небольших размерах страниц.

5.8.3. Интенсивность неудачных обращений за страницей и ее моделирование

Как неудачные обращения к буферу, так и неудачные обращения к основной памяти могут быть достаточно качественно промоделированы с помощью модифицированной модели независимых обращений [E2]. В соответствии с этой моделью если оказываются выбранными некоторый блок или страница, то существует большая вероятность того, что очередное обращение произойдет к тому же блоку или странице. Таким образом, оказывается, что новые блоки или страницы выбираются независимо с вероятностью, пропорциональной частоте их использования. При этом работа процессора, в ходе которой имеют место неудачные обращения, может быть аппроксимирована экспоненциальной моделью обслуживания, генерирующей усредненное количество запросов страниц в секунду со скоростью, которая пропорциональна интенсивности обращения процессора к памяти, умноженной на вероятность неудачного обращения. При работе типичного процессора на каждую команду приходится около двух обращений к памяти; следовательно, средняя интенсивность генерации обращений к страницам λ_ρ может быть дана в виде

$$\lambda_\rho = IPS \times M_{rp} \times 2(1+g)Fy,$$

где IPS — количество команд, обрабатываемых за секунду; M_{rp} — вероятность неудачного обращения за страницей; g — доля записанных страниц, подлежащих свопингу; Fy — доля времени, в течение которого процессор занят.

Чтобы мультипрограммный процессор был постоянно занят ($Fy \approx 1$), страничный диск должен быть в состоянии обслуживать запросы страниц со скоростью, большей или равной интенсивности выработки этих запросов. На рис. 5.29 иллюстрируется организация очереди к дисковой памяти, базирующаяся на построении отдельных очередей для каждого сектора и обслуживании заданий по принципу наименьшей латентности (т. е. задания, которые раньше достигают головок чтения/записи, обслуживаются в первую очередь) [F3]. В данном случае предполагается, что объем сектора равен одной странице.

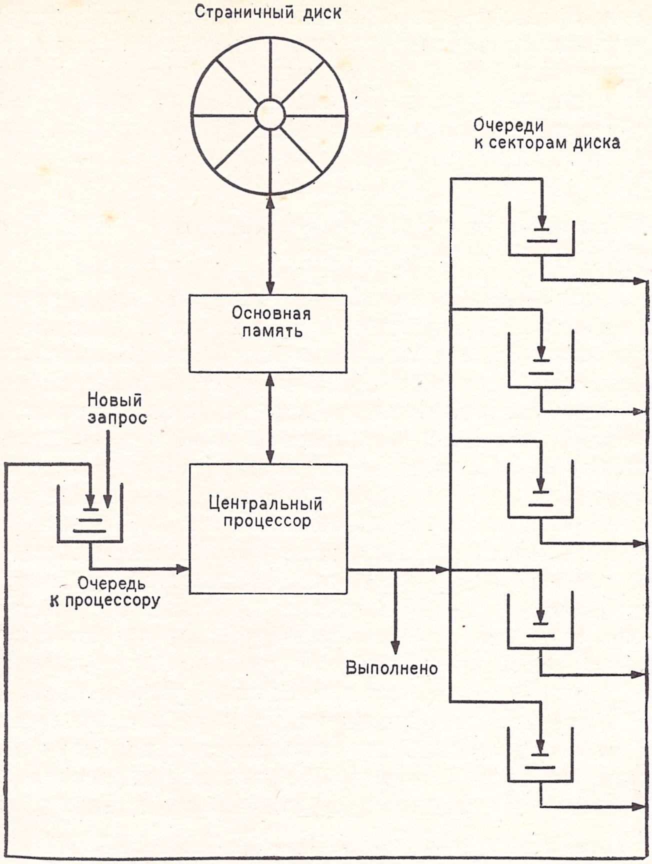


Рис. 5.29. Организация очереди к страничному диску.

5.8.4. Соотношение интенсивности запросов страниц и скорости их обслуживания

Рассмотрим два реальных примера. В одном из них будет страничная дисковая память, состоящая из 10 секторов, в каждом из которых содержится одна страница объемом 4096 байт. Будем считать, что диск вращается со скоростью 3600 об/мин и что всего имеется 400 дорожек. В виде обобщенных параметров эти величины выражаются следующим образом:

$$s\,(\text{количество секторов}) = 10,$$

$$tr\,(\text{задержка на оборот диска}) = 1/60 \text{ с}.$$

Если процессор обрабатывает одно задание, то при неудачном обращении за страницей он прекращает выполнение полезной работы и ожидает прибытия в основную память новой страницы. Среднее время, требуемое для обслуживания запроса страничным диском, определяется как

$$t_{\text{ср}} = (tr/2) + (tr/s).$$

Чтобы придать выполняемому анализу простоту и интуитивный характер, будем также считать дисковую память обслуживающим устройством экспоненциального типа, которое обеспечивает скорость обслуживания, обратную времени обслуживания. Очевидно, что диск не является экспоненциальным обслуживающим устройством, потому что осуществляемое им обслуживание никогда не длится менее tr/s с и никогда не превышает $tr(1 + 1/s)$ с. Более точный анализ потребовал бы учета распределения времени обслуживания, числа заданий, выполняемых в системе, и характеристик процессора как обслуживающего устройства.

Пусть используемые параметры имеют следующие числовые значения: $IPS = 2 \times 10^6$ с$^{-1}$, $M_{rp} = 10^{-4}$, $g = 0{,}25$. Тогда средняя интенсивность запросов страниц процессором

$$\lambda_\rho = 2 \times 10^6 \times 2 \times 1{,}25 \times 10^{-4}\, Fy = 500\, Fy/\text{с}.$$

Средняя скорость обслуживания для рассматриваемой дисковой памяти

$$\lambda_d = \frac{1}{t_{\text{ср}}}(1 - Fy) = \frac{600 \times 20}{60 \times 2 + 60 \times 10}(1 - Fy)/\text{с} =$$
$$= 100\,(1 - Fy)/\text{с}.$$

При сбалансированности системы скорости этих двух устройств обслуживания, описываемых экспоненциальными характеристиками, должны быть равны между собой:

$$500\, Fy = 100\,(1 - Fy), \qquad Fy = 1/6.$$

Таким образом, параметром, ограничивающим производительность системы, является обслуживающая способность диска, тогда как процессор занят только в течение одной шестой части рабочего времени.

Если процессор работает в мультипрограммном режиме, очереди к секторам дисковой памяти могут быть заполнены, в результате чего повысится скорость обслуживания процессора дисковой памятью и процессор будет пребывать в состоянии занятости относительно бóльшее время.

Для случая мультипрограммной работы скорость выполнения обслуживания страничным диском λ_{dm} может быть

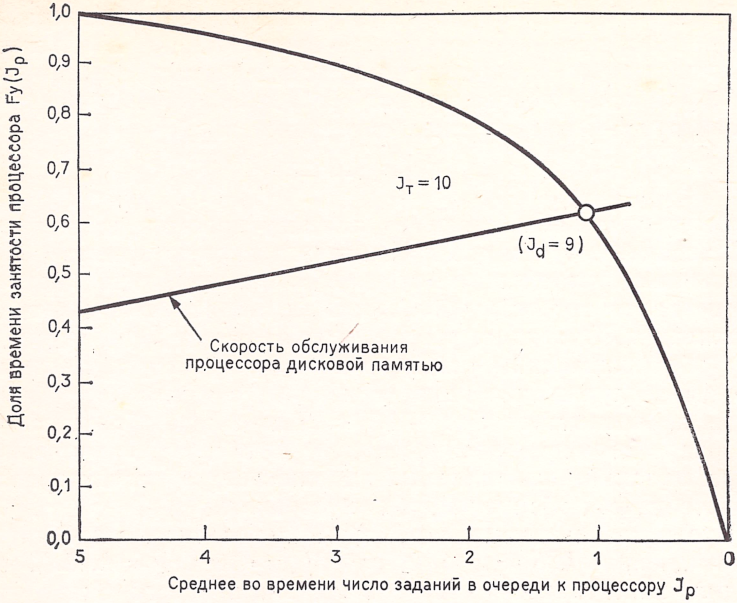


Рис. 5.30. Баланс очередей к дисковой памяти и процессору.

аппроксимирована следующим выражением [F3]:

$$\lambda_{dm} = \frac{s}{tr}\left[1 - (e)\exp - \frac{J_d + 1}{s + 1}\right],$$

причем $J_T = J_P + J_d$, т. е. суммарное число активных заданий равно сумме среднего во времени числа заданий в процессорной очереди и среднего во времени числа заданий в очередях секторов диска. Эта аппроксимирующая модель дает наилучшие результаты при $1 < J_d < 10$.

Вследствие того что процессор ведет себя как обслуживающее устройство экспоненциального типа, он не будет полностью занят, даже когда в его очереди в среднем находится одно задание. В некоторые моменты в ней не будет ни одного задания, а в другие — два или более. На рис. 5.30 иллюстрируется то обстоятельство, что для обеспечения занятости процессора в течение 90 % времени в очереди к нему должно находиться в среднем три-четыре задания. Очевидно, что при более точном анализе должен приниматься во внимание статистический характер как запросов на обслуживание, так и соответствующих ответов.

Если в очереди страничной дисковой памяти заносится очень большое число запросов, то каждый раз при подходе очередного сектора может быть обслужен какой-то запрос. Для дисковой страничной памяти, рассматриваемой в данном

примере, максимальная скорость составит при этом 600 (10×60) обслуживаний запросов в секунду. При меньшем, более часто встречающемся на практике количестве заданий интенсивность обслуживания снижается.

Предположим, что уровень мультипрограммирования процессора составляет 10. Процессор и дисковая память находятся в равновесии по времени, когда интенсивность запросов страниц и скорость предоставления страниц оказываются равны между собой:

$$IPS \times M_{rp} \times 2 \times (1+g)\,Fy\,(J_p) = \\ = \frac{s}{tr}\left[\left(1-(e)\exp-\frac{J_d+1}{s+1}\right)\right] \text{ в секунду,}$$

$$500Fy\,(1) = 600\left[\left(1-(e)\exp-\frac{9+1}{10+1}\right)\right] \text{ в секунду.}$$

Это равновесие в используемой упрощенной модели очереди наступает, когда в процессорной очереди находится одно задание, а в очередях дисковой памяти — девять заданий. Равновесие иллюстрируется точкой пересечения двух графиков на рис. 5.30; процессор занят при этом в течение 60 % времени.

Важно отметить, что в примере было использовано довольно малое значение вероятности неудачного обращения 10^{-4}. Оно может быть достигнуто лишь при отведении каждому заданию достаточно большого пространства основной памяти порядка 128 000 байт. Если бы процессор работал над интерактивными заданиями, при выполнении которых вероятность неудачного обращения за страницей была бы гораздо больше, например 5×10^{-4}, то процессор был бы занят в течение 15 % времени и почти все десять заданий постоянно пребывали бы в очередях дисковой памяти.

Технология изготовления запоминающих устройств, которые имели бы при разумной стоимости время обращения в диапазоне от 10 до 1000 мкс, была бы идеальной для подобных областей применения. В настоящее время эта задача решается путем использования основной памяти большой емкости и множества страничных дисков. При наличии памяти с промежуточными характеристиками оптимальная работа системы достигалась бы при меньших уровнях мультипрограммирования и емкостях основной памяти [P2].

5.8.5. Стратегии замещения страниц в основной памяти

Так как производительность системы в значительной степени зависит от вероятности неудачного обращения за

страницей, которая в свою очередь определяется емкостью памяти, отводимой заданиям, существует, таким образом, весьма мощный стимул для использования основной памяти с максимально возможной эффективностью. В процессе выполнения любой задачи необходимая емкость основной памяти все время изменяется, поэтому отведение заданию фиксированного объема памяти в ряде случаев приведет либо к ее нерациональному использованию, либо к нехватке для нужд конкретного задания. По этим причинам наряду с LRU были разработаны и другие стратегии замещения страниц в основной памяти.

При использовании стратегии LRU для задания отводится фиксированная емкость памяти, а когда возникает потребность в пространстве памяти, из него удаляется или разрушается новой записью страница, со времени последнего использования которой прошло наибольшее время. Очевидно, что для выявления такой страницы необходим обновляемый список страниц.

В случае применения стратегии рабочего множества [D1] для каждой страницы, относящейся к заданию, предусматривается, по существу, отдельный счетчик, содержимое которого увеличивается при каждом обращении к любой другой странице данного задания и становится нулевым при обращении к этой странице. Если значение счетчика превышает определенную величину τ, называемую параметром рабочего множества, страница становится доступной для удаления. Задания, характеризующиеся большими значениями вероятности неудачного обращения, могут воспользоваться бездействующим страничным пространством других заданий. Содержимое счетчиков, сопоставляемое страницам некоторого задания, получает положительное приращение только при запросах страниц, относящихся именно к этому заданию.

Стратегия удаления, основанная на частоте неудачных обращений за страницей [C5], учитывает число запросов, имеющих место между двумя неудачными обращениями. Если число таких запросов становится меньше определенной величины l, страницы, к которым имели место неудачные обращения, передаются в основную память, в результате чего интенсивность неудачных обращений снижается. В то же время, если это количество запросов превышает заданный порог, все страницы, которые находятся в основной памяти и к которым не было обращений с момента последнего неудачного обращения за страницей, становятся доступными для замещения их в основной памяти. Таким образом, данная стратегия обеспечивает быструю реакцию на изменение по-

требностей данного задания в использовании основной памяти.

В реальных системах рассмотренные стратегии часто используются в упрощенном виде. Кроме того, выбор между ними часто производится из соображений удовлетворения локальных требований.

УПРАЖНЕНИЯ

1. Сравните производительность систем памяти для случаев, когда один кэш емкостью 16К слов обслуживает одну память, содержащую 16 модулей, и когда 16 кэшей емкостью по 1К слов каждый обслуживает 16 модулей. Время цикла кэш-памяти положите равным 0,1 мкс, а время цикла модулей — 1 мкс; считайте, что применяется алгоритм простого свопинга с флагами ($X = 0{,}1$), а обращения к памяти происходят в произвольном порядке. Используйте значения вероятности удачного обращения, встречающиеся в данной главе.

2. Разработайте схему устройства, которое могло бы быть использовано в качестве коммутирующего переключателя для соединения четырех процессоров с восемью модулями памяти.

3. В трехуровневой иерархической организации памяти использованы запоминающие устройства с временами цикла 50 нс, 500 нс и 500 мкс. Стоимость в расчете на один бит в них составляет 0,1, 0,04 и 0,005 цента соответственно. Определите близкое к оптимальному соотношение ресурсов памяти первого и второго уровней.

4. В выражении для оптимального распределения ресурсов предполагается, что вероятность неудачного обращения определяется по формуле $M_r = a_1/(a_0 + W)$. Выведите выражение для распределения ресурсов в случае, когда $M_r = \alpha e^{-\alpha W}$.

5. Предположим, что 16 процессоров работают с использованием одной шины и интенсивность запросов на обслуживание от каждого процессора, если он занят, составляет 1/20 от пропускной способности шины. Определите долю времени, в течение которого шина занята, и долю времени, в течение которого занят каждый процессор. Для описания шины и процессора примените модели устройств обслуживания экспоненциального типа.

6. В рассмотренном примере проектирования системы с кэш-памятью в части кэша, предназначенной для размещения данных, были использованы ячейки памяти с общими линиями ввода-вывода, а для передачи данных в память и из нее во время операций записи и чтения служили сигналы выбора

кристалла. Определите, во сколько раз сократится время доступа, если использовать ячейки, аналогичные тем, которые применяются в указателе, а для связи запоминающих устройств с шиной вместо метода, основанного на выборе кристалла, применить вентили с временем переключения 6 нс.

7. Предположим, что процессор, фигурирующий в упр. 5, стоит 5000 долл., а шина и главная память стоят 50 000 долл. Увеличится или уменьшится отношение производительности системы к ее стоимости при введении в систему двух дополнительных процессоров.

Глава 6

Корректирующие коды и надежность быстродействующих систем памяти

6.1. ВВЕДЕНИЕ

Полупроводниковые запоминающие устройства в настоящее время широко применяются в самых разнообразных вычислительных системах и другой аппаратуре, имеющей отношение к вычислительной технике: больших ЭВМ, мини- и микроЭВМ, дополнительных устройствах хранения информации, периферийных усройствах и их контроллерах, удаленных терминалах и буферной памяти. Главной причиной широкого распространения этих устройств явилось достигаемое при их использовании существенное уменьшение отношения стоимости и производительности. На фоне постоянного снижения цен и повышения производительности надежность микросхем полупроводниковой памяти остается недостаточно высокой с точки зрения требований, предъявляемых к ним при создании больших вычислительных систем. Так как в составе подобных систем должно быть много ИС, надежность системы в целом при увеличении ее объема снижается. В связи с этим для повышения надежности таких систем применяется множество методов обнаружения и исправления ошибок, не связанных с большими затратами. Очевидно, что использование этих методов играет существенную роль и при построении больших систем памяти, базирующихся на иерархических принципах. В данной главе дан обзор методов обнаружения и исправления ошибок применительно к системам памяти с использованием кэшей.

6.2. КОДЫ, ОБЕСПЕЧИВАЮЩИЕ ОБНАРУЖЕНИЕ И ИСПРАВЛЕНИЕ ОШИБОК

Теория и методы технической реализации средств обнаружения и исправления ошибок хорошо разработаны. Использование этих средств применительно к полупроводниковым запоминающим устройствам позволяет разработчику реализовать низкую стоимость, высокую плотность компоновки, малое потребление мощности и высокое быстродействие при обеспечении достаточной надежности системы памяти. Наиболее часто в полупроводниковых системах памяти применяются коды, основу которых составляет код Хемминга, исправляющий одиночную ошибку [H1]. Коды, корректирующие

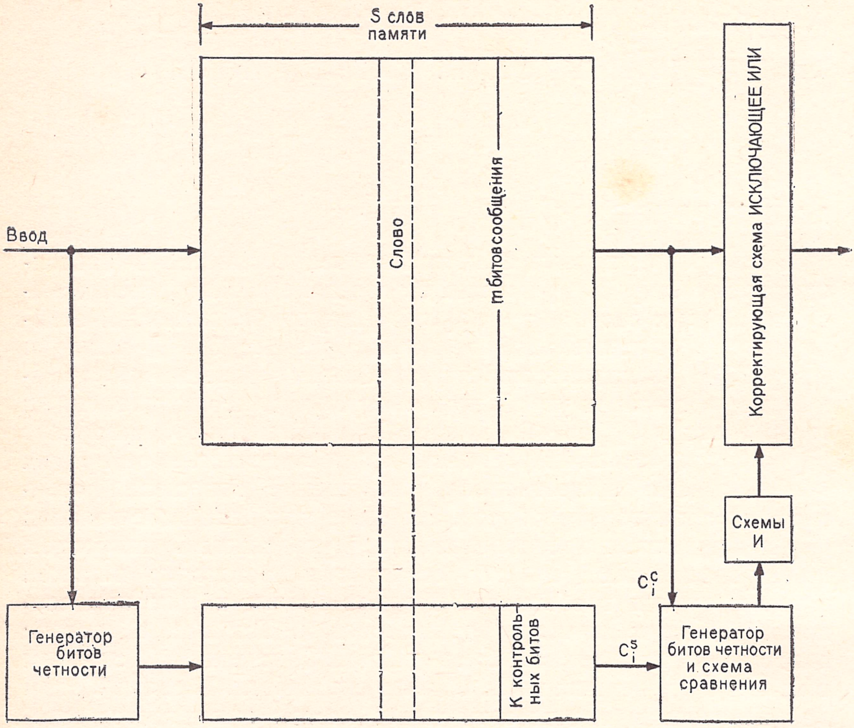


Рис. 6.1. Структурная схема памяти с обнаружением и исправлением ошибок.

множество ошибок, весьма сложны и поэтому в полупроводниковых запоминающих устройствах не находят широкого применения [P1].

Обнаружение и исправление ошибок осуществляются в общем случае путем добавления лишних (избыточных) битов к полю данных. На рис. 6.1 показана структура средств обнаружения и исправления ошибок в системе памяти. Из центрального процессора поступает m-разрядное слово, схема вырабатывает k бит контроля четности и в память заносится n-разрядное [т. е. $(m+k)$-разрядное] слово. Таким образом, k бит контроля четности присоединяются к информационным битам.

Все эти n бит хранятся в памяти, где они могут подвергнуться искажениям из-за неисправности компонент подсистемы памяти. При последующем считывании данных из памяти вырабатываются новые k бит контроля четности для тех же m информационных битов. Эти k вновь выработанных

бит сравниваются с k бит контроля четности, считываемыми из памяти, и в результате сравнения формируются k бит синдрома. Если нет ошибок, то все k бит синдрома являются нулями. Если произошло искажение информации, то в k бит синдрома содержатся сведения о том, в каком разряде данных имеет место ошибка, с тем чтобы эта ошибка могла быть исправлена и из памяти были переданы корректные данные; тем самым повышается надежность системы памяти.

Теория корректирующих кодов была глубоко разработана Лином [L8] и Питерсоном [P1]. Мы ограничимся здесь рассмотрением следующих трех видов корректирующих кодов:

коды, позволяющие обнаружить наличие одиночной ошибки;

коды с исправлением одиночной ошибки;

коды, позволяющие исправить одиночную ошибку и обнаружить наличие двойных ошибок.

Перед тем как приступить к анализу этих кодов, определим следующие термины.

(n, m)-код. Упорядоченный парный дескриптор кода, служащего для обнаружения и исправления ошибок, где n — полное число битов кодового слова, а m — число информационных битов, $k = n - m$ — число избыточных битов.

Коэффициент избыточности (R) — отношение полного числа битов (разрядов) кодового слова $(m + k)$ к числу разрядов исходного слова, не содержащего битов контроля четности, т. е. $R = (k + m)/m$.

Эффективность кода (*CE — Code efficiency*). Это еще одна мера избыточности данных. Она задается следующей формулой:

$$\text{CE} = m/n = m/(m + k)$$

в предположении, что все 2^m возможных сообщений являются равновероятными.

Кодовое слово. Элемент множества всех допустимых слов, определенных в кодовом пространстве. Кодовое пространство может быть определено как подгруппа, содержащая 2^m слов из числа тех 2^n слов, существование которых возможно в векторном пространстве кода длиной n бит. Слова, которые не являются кодовыми, определяются как слова с ошибками.

Расстояние. Минимальное различие в значениях битов двух кодовых слов. В литературе его называют также расстоянием Хемминга. Это расстояние можно определить как число битов, значения которых в сообщении надо изменить, чтобы образовалось другое сообщение.

6.2.1. Коды с обнаружением одиночных ошибок

С помощью этих кодов в переданной информации могут быть обнаружены одиночные ошибки. Для выявления одиночных ошибок широко применяется контроль по четности. Один из методов контроля по четности заключается в добавлении к информационному слову одного дополнительного бита. Если данные содержат $n-1$ бит, то в n-ю позицию помещается бит контроля четности, имеющий значение 0 или 1, в результате чего число единиц во всех n позициях оказывается либо четным, либо нечетным.

Значение 0 или 1 присваивается биту контроля четности в зависимости от того, нечетна или четна сумма единиц в битах слова данных, подлежащего контролю. Если полное число битов, имеющих значение 1, включая бит контроля четности, четно, то говорят, что слово данных четно; в противном случае слово данных принято считать нечетным. Обычно во всех подсистемах вычислительной системы для обеспечения единообразия используется один из этих вариантов контроля по четности.

Очевидно, что код, содержащий один бит контроля четности, представляет собой код для обнаружения одиночной ошибки, так как любое нечетное число ошибок при передаче искажает число единиц в кодовом слове.

Коэффициент избыточности кода, обнаруживающего одиночную ошибку при использовании одного бита контроля четности, имеет величину $R = n/(n-1)$. Видно, что для уменьшения избыточности n должно быть большим. Однако при увеличении n возрастает и вероятность появления ошибки. Кроме того, при использовании данного метода контроля по четности проверкой охватываются все биты кодового слова.

6.2.2. Коды, корректирующие одиночные ошибки

При использовании кодов Хемминга, служащих для исправления одиночных ошибок, к m информационным битам добавляется не один бит контроля четности, а k таких битов. Эти биты предназначены для контроля по четности в определенных разрядах m-разрядного информационного поля, их значения определяются путем кодирования.

Так как k бит синдрома должны указывать позицию бита, в котором имеет место одиночная ошибка, комбинация их значений должна описывать $m+k+1$ различных явлений. Таким образом, требования к размеру k-разрядного поля

определяются исходя из следующего соотношения:

$$2^k \geqslant m + k + 1,$$

или, если ввести обозначение $m + k = n$, то

$$2^m \leqslant 2^n/(n + 1).$$

На основании этого неравенства построена табл. 6.1, в которой приведены максимальные значения m и k для заданных n. Табл. 6.2 содержит значения коэффициента избыточности и эффективности кода для типичных форматов информационных слов.

Таблица 6.1. Длина кода и расстояние Хемминга

Число битов кода n	Число информ. битов m	Число контр. битов k	Число битов кода n	Число информ. битов m	Число контр. битов k
1	0	1	12	8	4
2	0	2	13	9	4
3	1	2	14	10	4
4	1	3	15	11	4
5	2	3	16	11	5
6	3	3	17	12	5
7	4	3	18	13	5
8	4	4	19	14	5
9	5	4	20	15	5
10	6	4	21	16	5
11	7	4	...	...	...

Таблица 6.2. Коэффициент избыточности и эффективность кода

	Число контрольных битов k				
	4	5	6	7	8
Число информац. битов m max $= 2^k - k - 1$	4—11	11—26	26—57	57—120	120—247
Полное число битов кодового слова $n = m + k$	8—15	16—31	32—63	64—127	128—255
Коэффициент избыточности $R = n/m$	$\frac{8}{4} - \frac{15}{11}$	$\frac{16}{11} - \frac{31}{26}$	$\frac{32}{26} - \frac{63}{57}$	$\frac{64}{57} - \frac{127}{120}$	$\frac{128}{120} - \frac{255}{247}$
Эффективность кода CE $= m/n$	$\frac{4}{8} - \frac{11}{15}$	$\frac{11}{16} - \frac{26}{31}$	$\frac{26}{32} - \frac{57}{63}$	$\frac{57}{64} - \frac{120}{127}$	$\frac{120}{128} - \frac{247}{255}$

Так как число битов, служащих для обнаружения ошибки, т. е. контрольных битов, k должно увеличиваться посредством единичных приращений, а добавление каждого такого бита обеспечивает обнаружение и исправление ошибок в диапазоне битов данных, коэффициент избыточности R и эффективность кода CE не являются непрерывными функциями от k. Из табл. 6.1 и 6.2 видно, что по мере увеличения длины слова коэффициент избыточности быстро убывает, а эффективность кода соответственно возрастает.

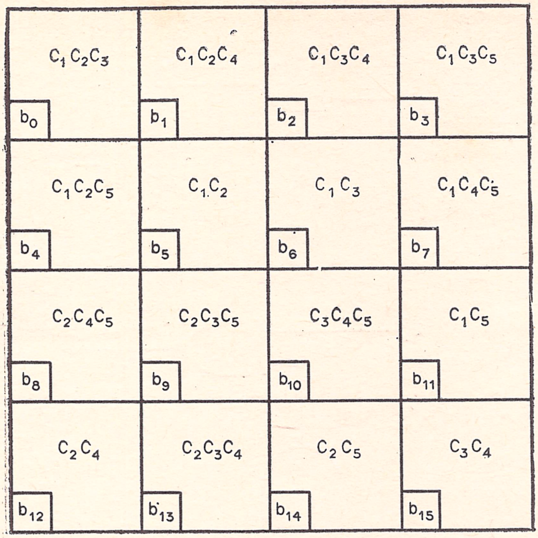
Коэффициент избыточности представляет собой весьма удобную характеристику возрастания затрат, имеющих место при введении в систему памяти средств обнаружения и исправления ошибок. Вследствие того что дополнительные биты должны храниться в памяти, коэффициент избыточности характеризует относительное увеличение стоимости памяти из-за необходимости хранить в ней контрольные биты. Кроме дополнительных затрат на память следует учитывать и расходы, определяемые наличием этих битов и связанные с построением генераторов битов четности, дешифраторов и компараторов, а также логических схем коррекции ошибок. Так как коэффициент избыточности относительно выше при меньших значениях длины слова, интуитивно можно предположить, что стоимость схемы обнаружения и исправления ошибок составляет относительно большую часть полной стоимости систем памяти, в которых слова имеют небольшую длину, и что эта часть стоимости уменьшается с увеличением длины информационных слов.

6.2.3. Размещение контрольных битов и схема коррекции

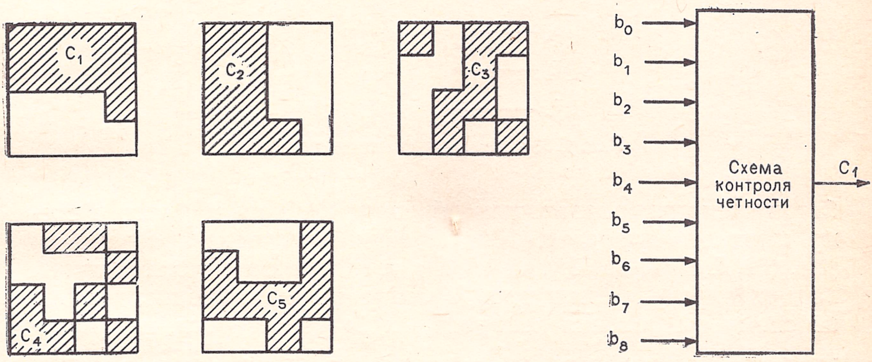
k бит четности служат для контроля определенных позиций из числа m информационных битов и в случае ошибки позволяют выработать информацию для коррекции. Обозначим биты контроля четности через $C_1, \ldots, C_k$, а информационные биты — через $I_1, \ldots, I_m$.

Контрольные биты могут быть размещены несколькими различными способами. Они могут быть разбросаны среди информационных битов или сосредоточены в конце слова. Столь же многими путями может быть организовано вычисление контрольных битов четности.

Если имела место ошибка в бите m-разрядного слова данных, то при обычном назначении два или более битов синдрома (ошибки) примут единичное значение. Если же ошибка произошла в значении одного из хранящихся в памяти битов четности, то при сравении его с вновь вычислен-



а



б

Рис. 6.2, *а* — сопоставление битов данных битам контроля четности; *б* — включение битов данных в схему вычисления различных контрольных битов четности (карты сопоставления).

ным значением этого бита единичное значение примет просто один бит синдрома.

На рис. 6.2, *а* приведен пример произвольного назначения, при котором каждый бит исходного 16-разрядного слова является источником входного сигнала для двух или трех из числа k схем контроля четности (C_i). Конкретный выбор

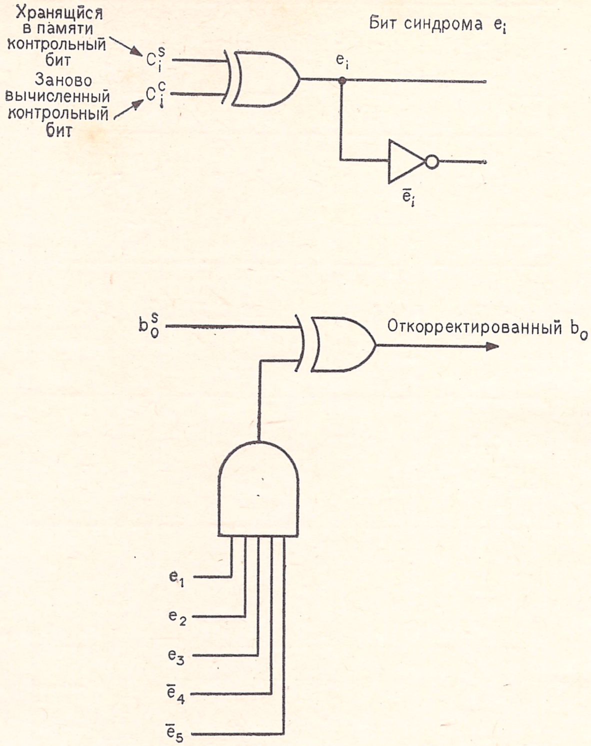


Рис. 6.3. Схема коррекции ошибок.

битов сделан произвольным образом; включение битов в схему вычисления контрольных значений четности показано на рис. 6.2, *б*. Каждый бит контроля четности формируется путем подачи определенного набора битов информационного слова на схему вычисления контрольных битов четности, как показано на рисунке. Например, для формирования бита контроля четности C_1 используются биты данных b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7 и b_{11}. Остальные биты контроля, т. е. биты четности, определяются другими комбинациями битов данных. С целью иллюстрации наложения полей контрольных битов 16-разрядное слово данных представлено в виде квадратной матрицы.

На рис. 6.3 показана схема, выполняющая коррекцию одиночных ошибок. В данном примере предполагается, что биты четности, т. е. контрольные биты, вычисляются с помощью набора пяти схем четности. Одна и та же схема определения четности может быть использована для генерации бита четности, при загрузке слова в память, а также при последую-

щем повторном определении контрольных битов с целью проверки. Для удобства контрольные биты, загружаемые в память, помечены на рис. 6.3 индексами s, а биты, вычисленные при повторном расчете, — индексами c.

Для получения каждого бита синдрома соответствующие биты контроля, хранящиеся в памяти и вычисленные заново, обрабатываются по схеме ИСКЛЮЧАЮЩЕЕ ИЛИ. Если их значения оказываются одинаковыми, выходной сигнал этой схемы (т. е. значение бита синдрома) оказывается равным 0, в противном случае бит синдрома равен 1. На рис. 6.3 биты синдрома обозначены символами e. Для коррекции какого-либо бита в исходном слове данных необходимо лишь воспользоваться этой имеющейся в распоряжении информацией. Если, например, имела место ошибка в бите b_0, биты синдрома 1, 2 и 3 будут иметь значение 1, а биты 4 и 5 — значение 0. При подаче на 5-входовую схему И сигналов e_1, e_2, e_3, $\bar{e}_4$ и $\bar{e}_5$ ее выходной сигнал однозначно укажет на наличие ошибки в бите 0. При обработке значения b_0, хранящегося в памяти, и выходного сигнала схемы И по схеме ИСКЛЮЧАЮЩЕЕ ИЛИ значение бита b_0 корректируется, т. е. возвращается к исходному правильному. Если выходной сигнал схемы И равен 0, значение бита не изменяется. Для каждого бита исходного слова данных требуется получить подобный управляющий сигнал и предусмотреть схему ИСКЛЮЧАЮЩЕЕ ИЛИ для коррекции результирующего значения бита в этом слове. Однако при этом не требуется значительного объема оборудования. Так как одна интегральная схема содержит, как правило, четыре или восемь схем ИСКЛЮЧАЮЩЕЕ ИЛИ или несколько схем И, общее количество ИС, необходимых для реализации описанной логики, оказывается довольно небольшим.

Справедливо также утверждение, что снижение производительности системы из-за включения в нее схемы коррекции ошибок незначительно, особенно если эта схема реализована на быстродействующих логических элементах. Если считать, что задержка схемы четности складывается из двух-трех единичных задержек элементов, то задержку схемы коррекции слова составляют пять-шесть задержек элементов. В случае применения, например, быстродействующих схем Шоттки или ЭСЛ увеличение времени обращения к памяти из-за схемы коррекции не превышает 10—30 нс. Для многих систем эта дополнительная задержка может быть несущественна.

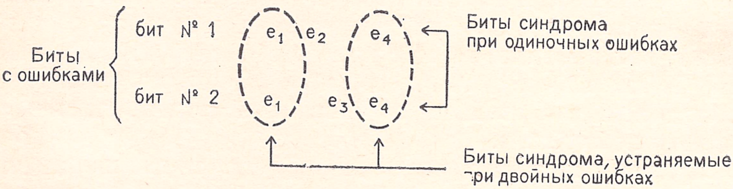
Отношение числа дополнительных разрядов, необходимых для коррекции ошибок, к числу битов слова данных все быстрее уменьшается по мере увеличения длины слова. По этой причине с точки зрения коррекции ошибок желательно строить память так, чтобы в ней хранились слова большой

длины. Если объем памяти более или менее значителен, то это достигается без труда.

Добавив к 16-разрядному слову данных, рассмотренному в примере, еще один контрольный разряд, можно обеспечить наряду с коррекцией одиночных ошибок обнаружение двойных. Это можно проиллюстрировать следующим образом.

Пусть каждый бит данных участвует в формировании трех из шести контрольных битов, причем каждому биту данных соответствует отличная от других комбинация контрольных битов. (Отметим, что количество сочетаний из 6 предметов по 3 без учета порядка их размещения равна 20, что больше требуемой величины 16.) При этом ошибка в одном бите данных вызовет появление единичного значения в трех битах синдрома, при ошибке же в контрольном бите значение логической 1 примет лишь один бит синдрома.

При возникновении ошибок в двух битах, участвующих в определении какой-либо четной суммы, бит синдрома не укажет на наличие ошибки, т. е. число битов синдрома, принимающих единичное значение, уменьшается. Таким образом, число битов синдрома, имеющих значение логической 1, при возникновении ошибок в двух битах оказывается четным. Как показано в приводимом ниже примере, оно может быть равно 2, 4 или 6, т. е. явным образом отличается от 3, что имеет место при одиночной ошибке в данных.



Ошибка в одном или двух контрольных битах также приводит к четному числу битов синдрома.

6.3. АНАЛИЗ НАДЕЖНОСТИ

Применение кодов, обеспечивающих обнаружение и исправление ошибок, значительно повышает достоверность данных, хранящихся в подсистеме памяти. В данном разделе мы количественно оценим это повышение качества посредством анализа надежности различных кодов, находящих применение в системах памяти. Сначала будут приведены основные положения традиционной теории надежности, а затем проанализировано влияние кодов с обнаружением и исправлением ошибок на надежность систем памяти.

6.3.1. Традиционная теория надежности

Надежность обычно определяется как способность компонент или оборудования успешно функционировать в течение некоторого времени t. Более конкретно это — вероятность сохранения работоспособности оборудования или системы в течение определенного интервала времени. Косвенным образом она характеризует вероятность появления отказов в системе за время ее нормального периода функционирования. Так как система состоит из некоторого количества компонент, надежность системы определяется надежностью ее элементов.

При классическом подходе к оценке надежности устанавливается функциональная зависимость между надежностью компонент и системы путем выявления влияния надежности компонент на надежность системы. В соответствии с этим подходом системы подразделяются на три категории.

Последовательная система. Если отказ любой компоненты может вызвать отказ всей системы, система относится к числу последовательных.

Параллельная система. Система считается параллельной, если ее отказ может иметь место только при отказе всех входящих в нее компонент.

Мажоритарная система. Это частный случай параллельной системы, в которой n компонент работают параллельно, но для сохранения работоспособности системы должны функционировать m из числа этих n компонент.

6.3.2. Общее уравнение надежности

Прежде чем попытаться оценить надежность системы в целом, необходимо сначала рассмотреть интенсивность отказов отдельных компонент. Предположим, что система состоит из большого (теоретически бесконечного) числа компонент и N этих компонент продолжают функционировать по истечении временного интервала t с некоторого произвольного момента времени. Очевидно, что с возрастанием t величина N будет уменьшаться, т. е. для всех $t \geqslant 0$ производная dN/dt имеет отрицательное или нулевое значение.

На этом основании можно заключить, что число отказов за единицу времени также зависит от t. Отношение скорости уменьшения N к N в любой момент времени t называется интенсивностью отказов; по отношению к времени t она обозначается следующим образом:

$$\lambda(t) = (dN/dt)/N.$$

Представим это уравнение в другом виде:

$$dN/N = -\lambda(t)\,dt.$$

Если считать, что интенсивность отказов $\lambda(t)$ представляет собой постоянную величину λ и не является функцией от t, то $dN/N = -\lambda dt$.

Интегрируя это выражение, получаем

$$\log N = -\lambda t + \text{константа}, \quad \text{или} \quad N = Ke^{-\lambda t},$$

где K — константа, а e — основание натурального логарифма (2,71828). Если $K = N_0$ (начальному значению N), то

$$N = N_0 e^{-\lambda t}.$$

Так как в момент функционирования t сохраняют работоспособность N компонент из исходного числа N_0, то вероятность того, что конкретный элемент не выходит из строя в течение времени t, определяется как

$$N/N_0 = e^{-\lambda t}.$$

Эту величину можно рассматривать как вероятность сохранения живучести системы, т. е. как характеристику ее надежности. Таким образом,

$$R(t) = e^{-\lambda t}.$$

Введя параметр $\tau = 1/\lambda$, представляющий собой среднее время безотказной работы, можно записать

$$R(t) = e^{-t/\tau}.$$

Это соотношение известно под названиями экспоненциального закона изменения надежности или экспоненциального закона изменения вероятности отказа.

На рис. 6.4 изображен график зависимости интенсивности отказов компоненты от времени работы. Его обычно называют «корытообразной» кривой. Отметим, что вероятность отказов имеет наибольшие значения на начальном этапе работы устройства, называемом «периодом детской смертности». В процессе полезной работы, т. е. в период нормального функционирования, вероятность отказов постоянна, что соответствует произвольному и независимому характеру отказов; лишь в течение этого периода можно оценить надежность системы, содержащей N компонент. Видно, что в дальнейшем интенсивность отказов снова возрастает по мере старения устройства, т. е. в зоне износа компонент, если таковая имеет место.

6.3.3. Основы теории распределения надежности

Если обозначить через $R(t)$ вероятность успешного функционирования, а через $F(t)$ вероятность отказа, то

$$R(t) \cup F(t) = 1 \quad \text{и} \quad R(t) \cap F(t) = 0.$$

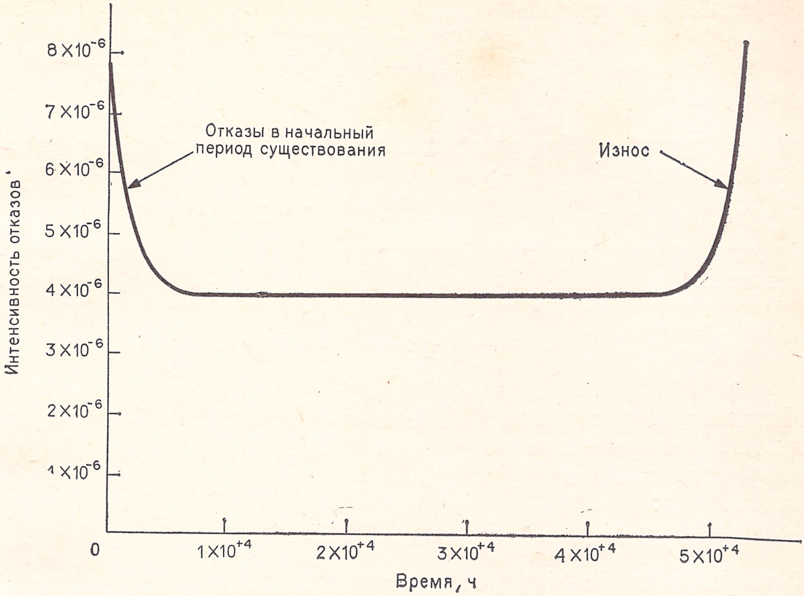


Рис. 6.4. Гипотетическая интенсивность отказов компонент.

Вероятность отказа $F(t)$ по существу может быть определена как $F(t)=\int_0^t f(t)\,dt$, где $f(t)$ — функция плотности отказов, а вероятность успешной работы определяется при этом как

$$R(t)=1-F(t)=1-\int_0^t f(t)\,dt.$$

Вероятность возникновения отказа в интервале времени между t_1 и t_2 можно рассчитать с помощью вероятностной функции

$$P=\int_{t_1}^{t_2} f(t)\,dt,$$

а так как время жизни всех устройств системы конечно,

$$P=\int_0^{\infty} f(t)\,dt=1.$$

На основании выражения

$$R(t) = 1 - \int_0^t f(t)\,dt$$

можно путем дифференцирования определить функцию плотности отказов

$$f(t) = dR(t)/dt.$$

Среднее время наработки на отказ (СВНО) задается выражением

$$\text{СВНО} = \int_0^\infty tf(t)\,dt.$$

Интенсивность отказов $\lambda(t)$ можно определить на основании любого распределения вероятностей как отношение функции плотности отказов $f(t)$ к функции, характеризующей надежность, $R(t)$:

$$\lambda(t) = f(t)/R(t).$$

Для устройств с экспоненциальным законом распределения отказов

$$R(t) = e^{-\lambda t}$$

функция плотности отказов

$$f(t) = \frac{dR(t)}{dt} = \lambda e^{-\lambda t},$$

$$\text{СВНО} = \int_0^\infty tf(t)\,dt = 1/\lambda.$$

Таким образом, СВНО для устройств, подчиняющихся экспоненциальному закону распределения отказов, есть величина, обратная интенсивности отказов. Этот параметр часто называют также *средним временем между отказами* (СВМО).

Интенсивность отказов устройств с экспоненциальной надежностью имеет, следовательно, вид

$$\lambda(t) = f(t)/R(t) = \lambda e^{-\lambda t}/e^{-\lambda t} = \lambda.$$

Очень часто на практике принимается, что надежность электронных компонент подчиняется экспоненциальным законам распределения. Одно из основных свойств экспоненциального распределения состоит в том, что отказы не зависят от времени. Это свойство позволяет варьировать сочетание числа устройств и количества часов их наработки как произведение. Так, например, если для испытания определенных компонент требуется 20 000 устройствочасов, то для выявления их на-

дежности можно испытывать либо 1000 компонент в течение 20 ч, либо 10 компонент в течение 2000 ч. Интенсивность отказов обычно поэтому выражается в процентной доле отказов устройств на 1000 ч наработки. Если интенсивность отказов является величиной постоянной, она может быть представлена следующими различными, но эквивалентными способами:

0,00001 отказ/ч = 10 отказ/10^6 ч = 0,1 %/100 ч = 1 %/1000 ч. При этом СВНО = 1/0,00001 = 100 000 ч.

Для определения надежности обобщенной системы необходимо сделать следующие предположения [B16]:

1. Все компоненты системы имеют постоянные интенсивности отказов, т. е. оборудование функционирует в течение полезного периода своего существования.
2. Отказы отдельных компонент не зависят друг от друга.
3. В распоряжении имеются данные по интенсивности отказов всех компонент, которые соответствуют реальным условиям их эксплуатации.
4. Чтобы система функционировала, должны быть работоспособны все ее компоненты.

6.3.4. Надежность последовательной системы

Если система содержит n компонент, функционально независимых друг от друга, то ее надежность

$$R(t) = \prod_{j=1}^{n} R_j(t),$$

где $R_j(t)$ — надежность j-й компоненты, т. е. надежность системы, представляет собой произведение надежностей ее отдельных компонент.

Если надежность этих компонент подчиняется экспоненциальному закону распределения отказов и их интенсивности отказов имеют величины $\lambda_1, \ldots, \lambda_n$, то надежность системы

$$R(t) = e^{-\lambda_1 t} e^{-\lambda_2 t} e^{-\lambda_3 t} \ldots e^{-\lambda_n t} = (e) \exp\left[-\sum_{j=1}^{n} \lambda_j t\right],$$

а СВНО последовательной системы

$$\text{СВНО} = \int_0^\infty t\,(e) \exp\left[-\left(\sum_{j=1}^{n} \lambda_j t\right) dt\right] = 1/\sum_{j=1}^{n} \lambda_j.$$

Если интенсивность отказов одинакова для всех компонент системы, то надежность системы определяется выражением

$$R(t) = e^{-n\lambda t}, \text{ а СВНО} = 1/n\lambda.$$

6.3.5. Надежность параллельной системы

Для того чтобы параллельная система перестала функционировать, должны выйти из строя все ее компоненты. Если опять предположить, что компоненты функционально независимы, то вероятность отказа системы будет определяться выражением

$$P(F) = \prod_{j=1}^{j=n} P(R_j),$$

где $P(R_j)$ — вероятность отказа j-й компоненты. Таким образом, если $Q(t)$ — ненадежность системы, а $Q_j(t)$ — ненадежность j-й компоненты, то

$$Q(t) = \prod_{j=1}^{j=n} Q_j(t),$$

а надежность системы

$$R(t) = 1 - Q(t) = 1 - \prod_{j=1}^{j=n} Q_j(t) = 1 - \prod_{j=1}^{j=n} [1 - R_j(t)].$$

Отсюда следует, что если ненадежность всех n компонент имеет экспоненциальный характер и их интенсивности отказов составляют $\lambda_1, \lambda_2, \ldots, \lambda_n$, причем они работают в системе параллельно, то выражение для надежности системы принимает вид

$$R(t) = 1 - \prod_{j=1}^{j=n} \left(1 - e^{-\lambda_j t}\right),$$

а если интенсивность отказов всех компонент одинакова и равна λ, то

$$R(t) = 1 - (1 - e^{-\lambda t})^n.$$

СВНО для параллельной системы описывается выражением

$$\int_0^\infty t [1 - (1 - e^{-\lambda t})^n] \, dt.$$

6.3.6. Надежность мажоритарной системы

В мажоритарной системе параллельно работают n компонент, но для обеспечения функционирования системы достаточно, чтобы m из них были в работоспособном состоянии. Таким образом, если вероятность успешного функционирования одного устройства

$$R(t) = e^{-\lambda t}, \text{ то } Q(t) = 1 - R(t) = 1 - e^{-\lambda t}.$$

Следовательно, вероятности успешного функционирования и отказа для всех комбинаций n устройств связаны биномиальным выражением $(R+Q)^n$:

$$(R+Q)^n = R^n + nR^{(n-1)}Q + \frac{n(n-1)}{2!}R^{(n-2)}Q^2 + \frac{n(n-1)(n-2)}{3!}R^{(n-3)}Q^3 + \ldots + Q^n = 1.$$

Первый член биномиального выражения $P_0 = R^n$ — это вероятность успешного функционирования всех устройств системы. Второй член $P_1 = nR^{n-1}Q$ — вероятность того, что отказало всего одно устройство, третий член $P_2 = [(n(n-1)/2!]R^{n-2}Q^2$ — вероятность одновременного отказа двух устройств и т. д. Последний член выражения $P_n = Q^n$ — вероятность отказа всех n устройств.

Обобщенное уравнение биномиального распределения имеет вид

$$R(t) = \sum_{i=m}^{n} R^m (1-R)^{n-m},$$

где $R = e^{-\lambda t}$ для компонент с экспоненциальными характеристиками надежности, а λ — интенсивность отказов.

$$\text{СВМО} = \int_0^\infty t \left[-\frac{d}{dt}\sum_{i=m}^{n} R^m (1-R)^{n-m}\right] dt.$$

6.4. ВЛИЯНИЕ КОРРЕКТИРУЮЩИХ КОДОВ НА НАДЕЖНОСТЬ СИСТЕМ ПАМЯТИ

Повышение надежности подсистем памяти, являющееся непосредственным результатом применения корректирующих кодов, было впервые проанализировано в 1966 г. Алленом [A5]. Аллен установил, что весьма существенную роль в реализации преимуществ корректирующих кодов играет физическая реализация элементов памяти. Он отметил, что организация традиционных магнитных запоминающих устройств не создавала предпосылок для простого применения корректирующих кодов. С точки зрения достижения наибольшей эффективности отказ одного устройства должен вызывать при чтении и записи ошибку не более чем в одном бите. В случае магнитной памяти отказ формирователя слова является причиной ошибок во многих разрядах слова. Аллен отметил, что для каждого бита слова должен использоваться отдельный блок ферритовой памяти. Таким образом, для хранения n-разрядного слова в памяти необходимо физически организовать n отдельных субблоков. При такой организации

каждый бит слова извлекается независимо от других из отдельного субблока памяти, снабженного собственными дешифраторами адреса, формирователями и усилителями считывания. Наиболее существенное преимущество подобной организации заключается в том, что последствия одиночной ошибки в ячейках, усилителях считывания и формирователях могут быть сведены к искажению не более чем одного бита информации. Вследствие физического разделения разрядов двойная ошибка в слове памяти определяет необходимость того, чтобы корректируемые ошибки возникали в двух субблоках памяти, состоящих из независимых групп компонент. Аллен высказал, кроме того, утверждение, что применение кодов, исправляющих одиночные ошибки, в системах с описанной организацией значительно повышает надежность памяти.

Организация полупроводниковых ЗУ обычно предполагает применение корректирующих кодов. В типичном случае полупроводниковые ИС памяти построены таким образом, что одна ИС содержит одноименные разряды группы слов. При этом отказ одной ИС не вызывает искажения двух или бóльшего числа битов одного слова.

6.4.1. Организация полупроводниковых ЗУ, содержащих схемы коррекции одиночных ошибок

Кристалл ИС полупроводниковой памяти обычно организован таким образом, что содержит m слов длиной n, причем $n = 1, 2, 4$ или 8 бит. С точки зрения коррекции ошибок предпочтительны слова длиной 1 бит. Если объем памяти составляет S бит, то для ее реализации требуется $k = S/m \cdot n$ кристаллов. Минимальное число слов, хранящихся в памяти, зависит от числа битов, составляющих емкость одной ИС. Когда, например, память строится из ИС ЗУ $4К \times 1$, а длина слова составляет 32 бит, то потребуются 32 интегральные схемы ЗУ $4К \times 1$, если каждое слово должно считываться или записываться в течение одного цикла. За исключением некоторых случаев применения микроЭВМ, память строится на гораздо большем числе ИС, нежели минимальное.

Рис. 6.5 позволяет легко установить среднее время наработки на отказ для запоминающих устройств, построенных полностью на интегральных схемах, объем которых лежит в широком диапазоне, через интенсивность отказов ИС в случае, когда коррекция ошибок не используется.

На рис. 6.5 представлена логарифмическая зависимость количества ИС памяти от среднего времени наработки на отказ, причем интенсивность отказов ИС используется в качестве параметра этой зависимости. Номограмма построена

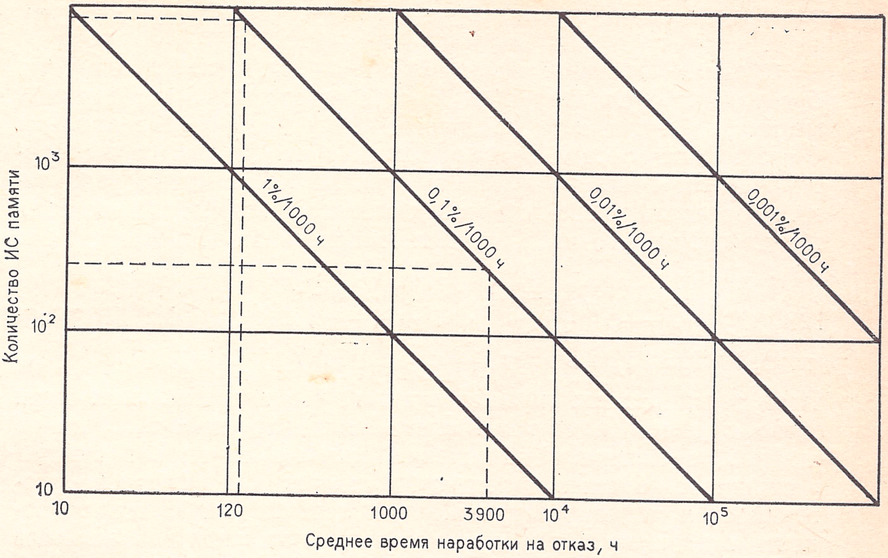


Рис. 6.5. Связь объема памяти, выраженного в количестве ЗУ, со средней наработкой на отказ.

на основе простого соотношения

$$\text{СВНО} = 1/d\lambda,$$

где d — количество ИС, λ — интенсивность отказов ИС.

Следует отметить, что при построении графиков приняты во внимание только ИС ЗУ с произвольной выборкой и не учтены другие компоненты, из которых состоит память. Графики лишь иллюстрируют изменение СВНО запоминающих устройств различного объема в зависимости от интенсивности отказов ИС памяти.

6.4.2. Повышение надежности с помощью кодов, исправляющих одиночные ошибки

Корректирующие коды позволяют повысить надежность запоминающих устройств и устранить ошибки, которые могут иметь место вследствие отказов ИС или искажения значений битов информации. Так, эксперименты показывают, что вероятность сбоев некоторых ИС динамических ЗУ емкостью 16К составляет 10^{-6} ч$^{-1}$ (10^{-6}/16 384 (бит·ч)$^{-1}$. Если память емкостью 1 млн. байт, организованная в виде 128К 64-разрядных слов, изготавливается с применением 512 таких ИС, то в предположении статистической независимости сбоев их

интенсивность для такой памяти составит $512 \cdot 10^{-6} = 5{,}1 \cdot 10^{-4}$ ч$^{-1}$.

Среднее время между неправильными срабатываниями будет иметь величину 2000 ч, т. е. около двух с половиной месяцев. Если такая же память была бы организована из 64-разрядных слов с 7 контрольными битами каждое и содержимое всех ячеек считывалось каждый час с целью устранения последствий сбоев, которые могли иметь место в течение предшествующего часа, то интенсивность отказов составляла бы примерно

$$\underset{\text{слова)}}{128\,000} \times \underset{\text{(2 из 71)}}{\frac{(71 \cdot 70)}{2}} \times \underset{\text{(2 ошибки)}}{\left(\frac{10^{-6}}{16384}\right)^2} \times \underset{\text{(без ошибок)}}{\left(1 - \frac{10^{-6}}{16384}\right)^{568 \times 16384}} \approx 1{,}2 \times 10^{-12} \text{ч}^{-1}$$

При этом расчете предполагается, что возникновение двух сбоев в одном и том же слове является основным видом отказов, а логика обнаружения ошибок не вносит вклада в снижение надежности. Если бы содержимое памяти не проверялось и не корректировалось в течение 100 ч, то вероятность отказа за время каждого часа в этом интервале была бы в 100 раз больше по сравнению со случаем, когда проверка осуществляется ежечасно. Обратите внимание, что в обоих этих случаях интенсивность отказов значительно ниже, чем при отсутствии избыточности.

Если ИС памяти имеют разрядную организацию, то одна и та же корректирующая схема может быть применена для устранения последствий как отказов ИС, так и случайных сбоев. Если бы интенсивность отказов ИС составляла 10^{-6}, то полная интенсивность отказов памяти, являющихся следствием отказов ИС, была бы равна

$$512 \cdot 10^{-6} \text{ч}^{-1} \approx 5 \cdot 10^{-4} \text{ч}^{-1}.$$

При добавлении средств коррекции, аналогичных рассмотренным выше, неисправные ИС заменялись бы в течение того же часа, когда произошел отказ, и интенсивность отказов памяти без учета отказов корректирующей логики составляла бы

$$\underset{\text{(столбцы)}}{8 \times \frac{71 \cdot 70}{2}} \quad \underset{\text{(2 неисправные ИС)}}{(10^{-6})^2} \quad \underset{\text{(исправные ИС)}}{(1 - 10^{-6})^{566}} = 2 \cdot 10^{-8} \text{ч}^{-1}.$$

В данном расчете отказы ИС опять считаются статистически независимыми; предполагается также, что для того, чтобы произошел отказ запоминающего устройства, необходимо, чтобы в двух отказавших ИС размещались биты одних и тех же слов. Случаи отказа трех или большего количества ИС не рассматриваются. Обратите внимание и здесь на существенное снижение интенсивности отказов. Это преимущество прояв-

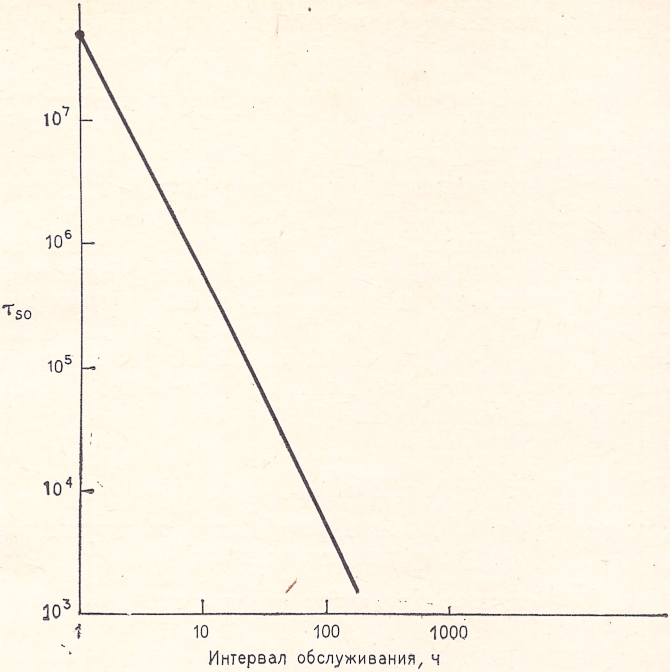


Рис. 6.6. Зависимость времени наработки до достижения вероятностью отказа значения 50 % (T_{50}) от интервала обслуживания системы (568 ИС памяти, интенсивность отказов каждой ИС 10^{-6} ч$^{-1}$, коррекция одиночных ошибок, для отказа системы необходим отказ двух ИС в одном столбце).

ляется в меньшей степени, если неисправные ИС в системе не заменяются, а накапливаются. При отсутствии обслуживания вероятность безотказной работы данной избыточной системы в течение 2000 ч составила бы 50 %. Для неизбыточной системы вероятность безотказной работы снизилась бы до 50 % за 1400 ч функционирования.

Если считать, что период обслуживания системы гораздо меньше среднего времени наработки на отказ, то среднее время наработки до достижения вероятностью отказа значения 50 % (TM_{50}) может быть определено следующим образом. Пусть b_m — интервал обслуживания. Тогда

$$P\begin{pmatrix}\text{все в порядке}\\ \text{в } b_m\end{pmatrix}=P\begin{pmatrix}\text{нет отказов}\\ \text{в } b_m\end{pmatrix}+P\begin{pmatrix}\text{отказ одной ИС}\\ \text{в } b_m\end{pmatrix}+$$

$$+P\begin{pmatrix}\text{два отказа не в}\\ \text{одном столбце}\\ \text{в } b_m\end{pmatrix}; \quad P\begin{pmatrix}\text{все в порядке}\\ \text{в } b_m\end{pmatrix}^s\approx 0{,}5,$$

$$TM_{50}=s\cdot b_m.$$

На рис. 6.6 иллюстрируется уменьшение среднего времени наработки на отказ при увеличении интервала обслуживания.

6.5. СЕРИЙНО ВЫПУСКАЕМЫЕ СХЕМЫ КОРРЕКЦИИ ОДИНОЧНЫХ ОШИБОК

Ряд фирм-изготовителей полупроводниковых компонент выпускает ИС, реализующие функции коррекции одиночных ошибок в запоминающих устройствах. Ниже приводятся технические характеристики ИС AM2960 фирмы Advanced Micro Devices. Она организована таким образом, что одна или несколько таких ИС могут быть применены для ЗУ с длиной слов 16, 32, 48 и 64 бит. Пользуясь бóльшим количеством контрольных битов, чем это необходимо, можно обеспечить и коррекцию двойных ошибок.

Am2960

Наращиваемое 16-битовое устройство обнаружения и исправления ошибок

ОБНОВЛЕННЫЕ ДАННЫЕ

Основные характеристики

- Модифицированный код Хемминга
 Обнаруживает многократные и корректирует одиночные ошибки в параллельном слове данных. Идеален для систем динамической памяти.

- Наращиваемость
 Одна ИС Am2960 — обнаружение и коррекция ошибок в 16-разрядных словах.
 Две ИС Am2960 — обнаружение и коррекция ошибок в 32-разрядных словах.
 Четыре ИС Am2960 — обнаружение и коррекция ошибок в 64-разрядных словах.

- Выработка синдромов
 При появлении ошибки Am2960 выдает биты синдрома, позволяющие определить место возникновения ошибок в памяти.

- Совместимость с микропроцессором
 ИС Am2960 ориентирована на применение в системах на основе микропроцессора Z8000, а также в высокопроизводительных системах серии 2900.

- Передовая технология изготовления и функционирования
 Новейшие методы изготовления БИС серии 2900, обеспечивающие высокую производительность.

Типичное значение промежутка времени от поступления данных до обнаружения ошибки 30 нс.

Типичное значение промежутка времени от поступления данных до коррекции 50 нс.

- Встроенная диагностика

 Дополнительные логические схемы в составе ИС обеспечивают реализацию диагностических функций в процессе контроля ИС и в ходе диагностики системы.

Общее описание

Устройство обнаружения и исправления ошибок Am2960 содержит логические схемы, необходимые для генерации контрольных битов для 16-разрядного поля данных в соответствии с модифицированным кодом Хемминга и для коррекции слова данных при выдаче контрольных битов. Обрабатывая данные, считываемые из памяти, Am2960 исправляет любую одиночную ошибку, обнаруживает все двойные и некоторые тройные ошибки. При работе с 16-разрядными словами используются 6 контрольных бит. ИС Am2960 могут соединяться для обработки 32-разрядных слов (7 контрольных бит) и 64-разрядных слов (8 контрольных бит). Во всех конфигурациях на специальные выводы ИС выдается синдром ошибки для коррекции данных.

Am2960 имеет два режима диагностики, в соответствии с которыми диагностические данные могут быть поданы на отдельные части кристалла для упрощения контроля ИС и выполнения диагностических функций системы. Изделие поставляется в герметичном корпусе DIP с 46 внешними выводами.

Архитектура устройства обнаружения и исправления ошибок

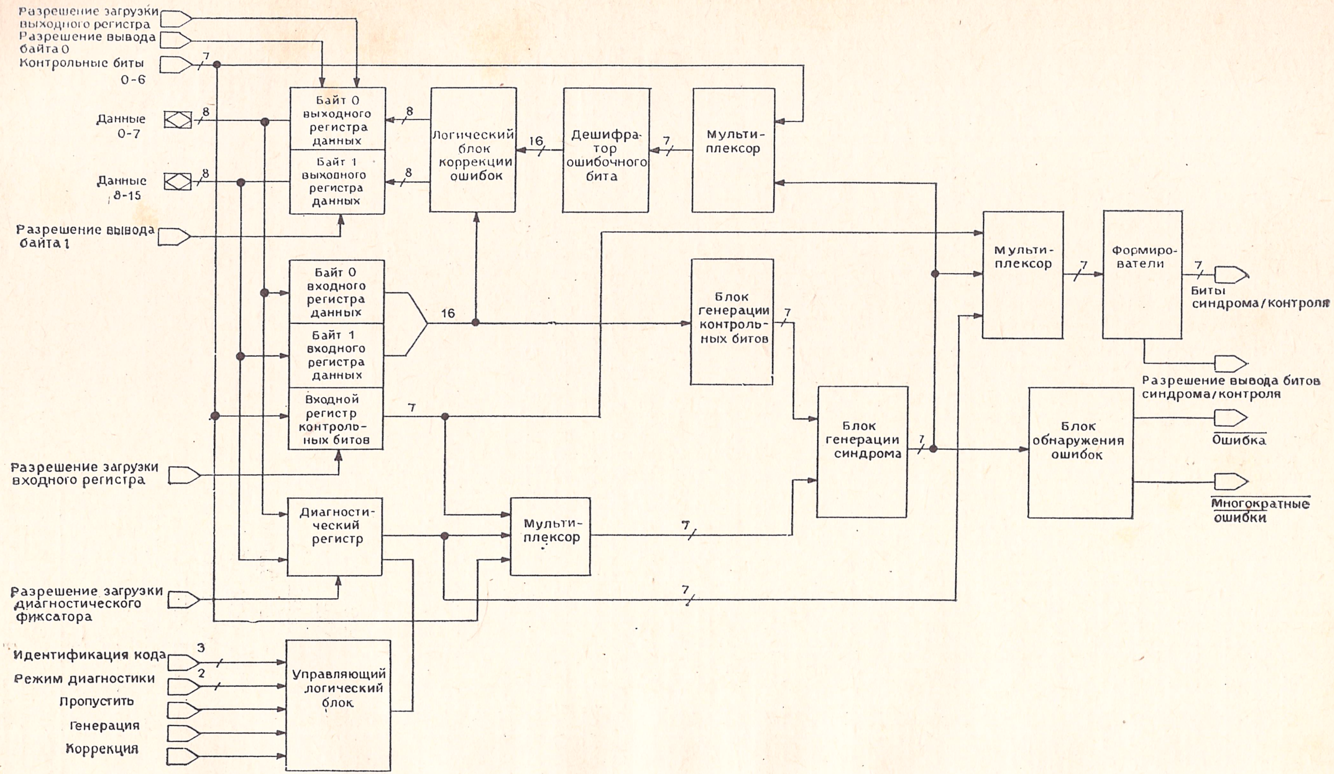
Am2960 представляет собой мощное 16-битовое разрядно-модульное устройство, служащее для генерации контрольных битов, обнаружения и исправления ошибок и диагностики.

Как видно из структурной схемы, устройство содержит следующие блоки: входной регистр данных, входной регистр контрольных битов, логический блок генерации контрольных битов, логический блок генерации синдрома, логический блок обнаружения ошибок, логический блок коррекции ошибок, выходной регистр данных, диагностический регистр, управляющий логический блок.

Входной регистр данных

16 бит данных загружаются с двунаправленных линий ДАННЫЕ при подаче входного управляющего сигнала

Структурная схема



(Сигнал разрешения вывода битов синдрома/контроля поступает на формирователи).

«Разрешение загрузки входного регистра» (LE IN). В зависимости от режима управления входные данные используются либо для генерации контрольных битов, либо для обнаружения/исправления ошибок.

Входной регистр контрольных битов

Семь контрольных битов загружаются управляющим сигналом LE IN. Контрольные биты используются в режимах обнаружения ошибок и исправления ошибок.

Логический блок генерации контрольных битов

Этот блок вырабатывает контрольные биты для 16-разрядных данных, находящихся во входном регистре данных. Контрольные биты генерируются в соответствии с модифицированным кодом Хемминга.

Логический блок генерации синдрома

Как в режиме обнаружения, так и в режиме исправления ошибок этот логический блок сравнивает контрольные биты, считанные из памяти, с набором контрольных битов, заново генерированным для считанных из памяти данных. Если оба набора битов оказываются одинаковыми, то это свидетельствует о том, что ошибок нет. Если имеются расхождения, значит, присутствуют одна или несколько ошибок в битах данных или контрольных битах.

Биты синдрома вырабатываются путем выполнения операции ИСКЛЮЧАЮЩЕЕ ИЛИ над двумя наборами контрольных битов. Если эти наборы идентичны (т. е. ошибок нет), все биты синдрома оказываются равными нулю. Если же ошибки есть, то комбинация битов синдрома может быть декодирована с целью определения числа ошибок и выявления бита, в котором имеет место ошибка.

Логический блок обнаружения ошибок

Этот блок декодирует биты синдрома, выработанные логическим блоком генерации синдрома. Если ошибок ни в данных, ни в контрольных битах нет, то выходные сигналы ОШИБКА и МНОГОКРАТНЫЕ ОШИБКИ сохраняют высокий уровень. При обнаружении одной или нескольких ошибок сигнал ОШИБКА имеет низкий уровень. При обнаружении двух или бóльшего числа ошибок оба сигнала ОШИБКА и МНОГОКРАТНЫЕ ОШИБКИ имеют низкий уровень.

Логический блок коррекции ошибок

В случае одиночных ошибок логический блок коррекции осуществляет инверсию (коррекцию) ошибочного бита. Откорректированные данные загружаются в выходной регистр

данных, из которого информация может быть затем считана на двунаправленные линии данных. Если одиночная ошибка имеет место в одном из контрольных битов, то с логического блока коррекции подаются откорректированные контрольные биты на выходы синдрома/контроля. Если откорректированные контрольные биты необходимы, устройство может быть переключено в режим генерации.

Выходной регистр данных

Выходной регистр данных служит для хранения результатов операции коррекции. Информация в регистр загружается из логического блока коррекции ошибок под управлением сигнала «Разрешение загрузки выходного регистра данных» (LE OUT). Этот регистр также может быть загружен непосредственно из входного регистра данных под управлением входного управляющего сигнала «Пропустить» (PASS THRU). Выходной регистр данных подразделяется на два 8-разрядных (байтовых) регистра, на которые могут быть поданы два независимых сигнала, разрешающие считывание их содержимого на двунаправленные линии данных.

Диагностический регистр

Это 16-разрядный регистр, загрузка которого производится с двунаправленных линий данных под управлением сигнала «Разрешение загрузки диагностического регистра» (LE DIAG). В одном байте диагностического регистра содержатся контрольные биты, а в другом — управляющая информация. Диагностический регистр служит либо для управления устройством при работе в режиме внутреннего управ-

Таблица 6.3. Схема формирования контрольных битов 16-разрядного модифицированного кода Хемминга

Генерируемые контрольные биты	Четность	Участвующие биты данных															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CX	Чет. (ИСКЛ. ИЛИ)		×	×	×		×			×	×		×			×	
C0	Чет. (ИСКЛ. ИЛИ)	×	×	×		×		×		×		×		×			
C1	Нечет. (ИСКЛ. ИЛИ—НЕ)	×			×	×			×		×	×			×		×
C2	Нечет. (ИСКЛ. ИЛИ—НЕ)	×	×				×	×	×				×	×	×		
C4	Чет. (ИСКЛ. ИЛИ)			×	×	×	×	×	×							×	×
C8	Чет. (ИСКЛ. ИЛИ)									×	×	×	×	×	×	×	×

Контрольный бит генерируется в результате выполнения операции ИСКЛЮЧАЮЩЕЕ ИЛИ или ИСКЛЮЧАЮЩЕЕ ИЛИ-НЕ над восемью битами данных, обозначенными в таблице символом «×».

ления, либо для выдачи контрольных битов при работе в одном из диагностических режимов.

Управляющий логический блок

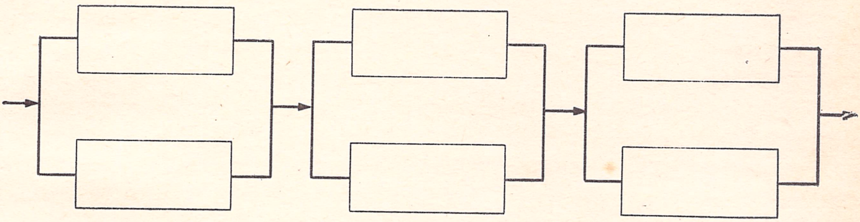
Управляющий логический блок определяет конкретный режим, в котором функционирует устройство. Обычно его работой управляют внешние управляющие сигналы. Однако в режиме внутреннего управления управляющие сигналы подаются от диагностического регистра.

УПРАЖНЕНИЯ

1. Длина слов памяти составляет 120 бит. Сколько контрольных битов требуется для реализации исправления одиночных ошибок?

2. Длина слов памяти составляет 40 бит. Определите количество контрольных битов, необходимое для обеспечения исправления одиночных ошибок и обнаружения двойных.

3. В комнате имеются четыре лампы. Среднее время наработки на отказ одной лампы составляет 1000 ч. Какова вероятность того, что после 500 ч будут продолжать работать две или более ламп?



4. Дублированная избыточная система имеет показанную на рисунке последовательно-параллельную организацию. Определите вероятность сохранения работоспособности системы как функцию времени; каждый блок имеет интенсивность отказов 10^{-6} ч$^{-1}$. Для того чтобы система работала, необходимо наличие сквозного пути в схеме.

5. Память, содержащая 48-разрядные слова, должна быть снабжена схемой обнаружения двойных ошибок. Необходимо использовать 7 контрольных битов. Сопоставьте 7 контрольных битов 48 битам данных. Разработайте логику указания двойных ошибок.

6. Рассмотрите память, в составе которой используются ИС емкостью 4096×1 и которая организована в виде 65 536 слов длиной $48 + (7)$ бит. Определите вероятность отказа после 1000 ч функционирования для случаев наличия

и отсутствия средств коррекции ошибок. Считайте, что интенсивность отказов ИС равна 2×10^{-7}, а для того чтобы память отказала, должны отказать две ИС в одном и том же столбце. Интенсивность отказов логических схем коррекции не учитывайте.

7. Коррекция одиночных ошибок с увеличением длины слова становится более эффективной. Определите эффективность 1024-разрядного слова с коррекцией ошибок.

8. Выполните задачу 6, считая, что ИС с одиночными отказами заменяются каждые 100 ч. Определите вероятность отказа системы по истечении 1000 ч.

Эволюция иерархических систем памяти и систем с кэш-памятью

7.1. ИСТОРИЯ РАЗВИТИЯ ИЕРАРХИЧЕСКИХ СИСТЕМ ПАМЯТИ

Идея использования иерархии не доступной для пользователя памяти с целью согласования по быстродействию центрального процессора и запоминающих устройств, которая дает пользователю впечатление использования бесконечной памяти, известна уже давно. Первая универсальная электронная цифровая вычислительная интегрирующая машина ENIAC, созданная в Пенсильванском университете, имела простую иерархию. Она содержала 20 промежуточных запоминающих регистров, выполненных на электронно-лучевых трубках для десяти десятичных цифр, и перенастраиваемую постоянную память емкостью 312 слов, причем для хранения промежуточных результатов использовались перфокарты. Скорость вычислений ограничивалась здесь скоростью перфорации. Еще в 1947 г. в Манчестерском университете в Англии была создана ЭВМ первого поколения, использующая быстродействующие индексные регистры, запоминающее устройство на вакуумных трубках Уильямса и вспомогательную память на магнитных барабанах. И основная, и вспомогательная память были разбиты на страницы фиксированного объема, которые передавались в электростатическую память ЭВМ во время выполнения программы.

В 1948 г. в ЭВМ с селективной выборкой (SSEC) фирмы IBM также была иерархическая система памяти, состоящая из перфокарточной памяти емкостью 20 000 слов, релейной памяти емкостью 150 слов и электронной памяти емкостью восемь слов. Каждое слово имело 19 разрядов и знак [B13]. Фактически SSEC была первой программируемой ЭВМ фирмы IBM, которая могла трактовать свои команды как данные, модифицируя их и воздействуя на результат [A7].

В 1956 г. фирма Ferranti Mercury [L5] использовала в своей ЭВМ иерархическую систему памяти, содержащую небольшую ферритовую память емкостью 40 Кбит, дополненную памятью на магнитном барабане емкостью 640 Кбит. Задача управления памятью этой системы целиком возлагалась на программистов, которые должны были вводить оверлейные программы при организации любого обмена между различными уровнями этой памяти. Было обнаружено, что из-за

того, что эффективное использование системы памяти требует детального понимания ее работы, только опытные программисты могли эффективно использовать систему, тогда как остальные тратили бóльшую часть машинного времени на неэффективную передачу информации с одного уровня памяти на другой.

Брукером [B15] была предложена удобная интерпретирующая система программирования, позволяющая работать с двухуровневой системой памяти, как с одноуровневой. Однако ее использование вело к существенной потере быстродействия.

В 1958 г. Бухгольц [B12] предложил схему двухуровневой иерархии памяти, состоящую из быстродействующей памяти емкостью 512 слов (время выборки 0,5 мкс) и большой медленнодействующей памяти второго уровня иерархии (емкостью 8K, блоки ферритовой памяти с временем выборки 12 мкс) для ЭВМ STRETCH фирмы IBM. Целью разработки этой ЭВМ было улучшить характеристики по сравнению с тогда существовавшей ЭВМ 704 на два порядка. Для этого процессор был спроектирован в виде двух асинхронных устройств: устройства выборки команды и устройства выполнения команды с отдельными запоминающими устройствами для команд и операндов. После разделения памяти между двумя асинхронными устройствами, доступ к которым осуществлялся независимо, и размещения команд и операндов в разных запоминающих устройствах стало возможным осуществить совмещение циклов выборки команды и операнда и добиться высокой производительности.

В системе Larc-1 фирмы UNIVAC [E3, E4] также была использована простая иерархическая система памяти, состоящая из основной памяти (с циклом 4 мкс) и буфера (с циклом 1 мкс) и сложной конвейерной схемы, обеспечивающей одновременную обработку четырех макрокоманд. Одновременно с выборкой команды I_n осуществлялась модификация (индексация) команды I_{n-1}, производилась выборка операнда для команды I_{n-2} и выполнялась команда I_{n-3}. Логическая схема, управляющая совмещением во времени упорядоченного выполнения команд на основе принципа конвейерной обработки, была достаточно сложной. Наряду с возможностью временнóго совмещения обращений к памяти в системе Larc-1 были предусмотрены 99 регистров сверхоперативной памяти, позволяющие уменьшить число таких обращений.

Схемы организации памяти этих машин первого поколения оказали огромное влияние на ЭВМ второго поколения, начиная с машины ICL ATLAS.

Система виртуальной памяти ЭВМ ATLAS фирмы ICL

Хотя идея виртуальной памяти, которая дает пользователю впечатление применения бесконечной памяти и автоматизирует все задачи управления памятью, была известна уже давно, впервые она была реализована только в 1961 г. ЭВМ ATLAS была первым удачным воплощением «полностью автоматизированной, простой для пользователя одноуровневой системы виртуальной памяти» [K8]. Хотя в этой системе и использовалась иерархия памяти, пользователю она представлялась в виде единой одноуровневой системы памяти.

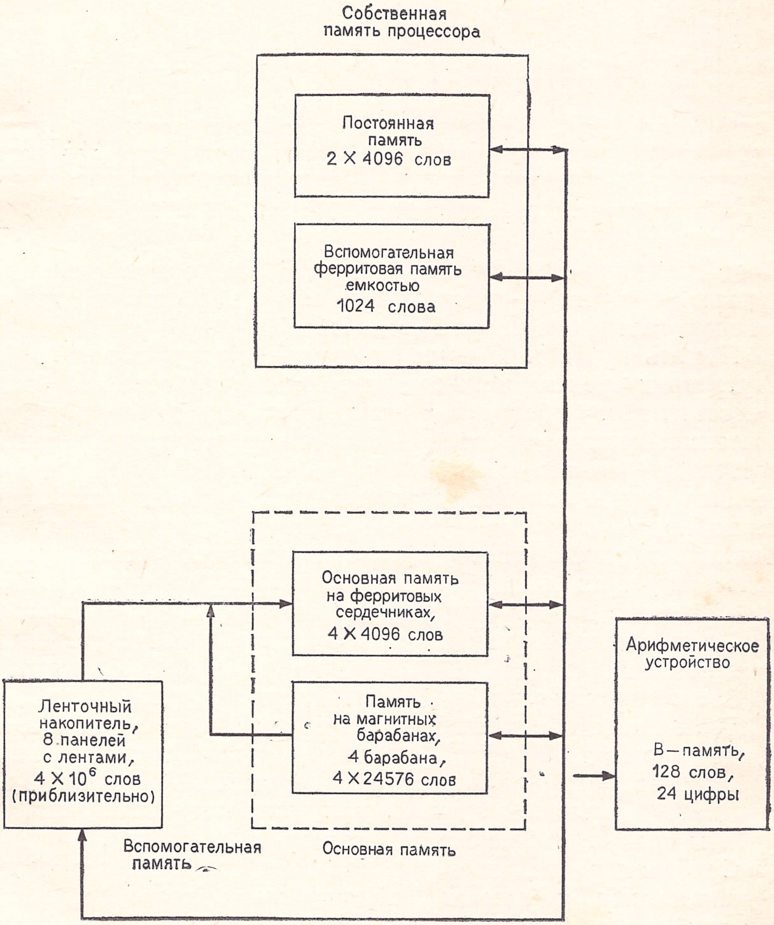


Рис. 7.1. Система иерархии памяти ЭВМ ATLAS.

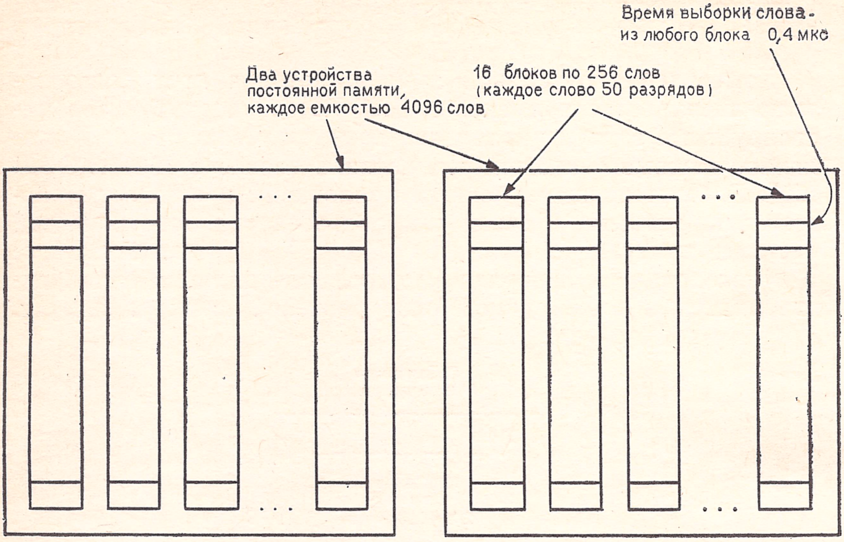


Рис. 7.2. Постоянная памяти ЭВМ ATLAS.

В действительности же ЭВМ ATLAS имела трехуровневую иерархию памяти — память процессора, основную память и вспомогательную память (рис. 7.1).

Память процессора была полностью доступна только для внутренних средств машины, для пользователя ее как бы не существовало. Собственная память состояла из двух частей: постоянной памяти и вспомогательной памяти. Концепция постоянной памяти, использованной в машине ATLAS, аналогична концепции микропрограммной памяти, распространенной в современных ЭВМ. Постоянная память выполняла функции запоминающего устройства для часто используемых математических функций, программ преобразования, устройств ввода-вывода и программ управления памятью. Эта постоянная, или односторонняя, память состояла из двух устройств; каждое имело емкость 4096 слов длиной 50 бит, разбитых на 16 блоков по 256 слов (рис. 7.2). Время выборки слова в отдельном блоке составляло 0,4 мкс, а выборка слова с изменением номера блока занимала 1,4 мкс. Вспомогательная ферритовая память процессора содержала 1024 ячейки для временного хранения информации, что было достаточно для работы программ, записанных в постоянной памяти.

Основная память представляла собой комбинацию ферритовой памяти и памяти на магнитных барабанах, общая емкость которой составляла 10^6 слов. Ферритовая память была организована в виде четырех блоков, каждый емкостью

4096 слов, память на четырех магнитных барабанах имела емкость 96 000 слов. И ферритовая, и память на барабанах разделены на страницы фиксированного объема в 512 слов; такая страница была основной единицей обмена информацией между ферритовой памятью и памятью на барабанах. Магнитные ленты обеспечивали очень большой дополнительный объем памяти (4 млн. слов). Обмен данными между управляющей памятью и лентой осуществлялся также блоками длиной 512 слов.

Адрес основной ферритовой памяти задавался либо центральным процессором, либо памятью на барабанах или ленте. Доступ к ферритовой памяти обеспечивался с заданными приоритетами, причем память на барабанах имела наивысший приоритет, затем следовала магнитная лента, процессор имел наинизший приоритет.

Организация ферритовой памяти

Ферритовая память была разделена на четыре независимых устройства, каждое из которых имело собственный дешифратор адреса и механизм чтения-записи. Допускался одновременный выбор нескольких устройств. Устройства объединялись в две пары. В одной паре находилась одна половина адресуемой ферритовой памяти, а в другой — вторая. Эти пары устройств некоторым образом чередовались. Смежные физические адреса находились в разных устройствах, что позволяло выбирать две команды в одном цикле обращения к памяти по единственному запросу памяти. Чтобы минимизировать число обращений к памяти и улучшить выборку по равнозначным адресам, были введены пары командных и адресных регистров.

Была сделана очень важная попытка сгладить различие в скоростях работы центрального процессора и ферритовой памяти. Для этого была введена очень быстрая регистровая память, состоящая из 128 индексных регистров (так называемых В-регистров), из которых процессор мог выбрать операнд половинной длины за время, сравнимое со скоростью работы процессора. Эти регистры имели целевое назначение: некоторые применялись в системных программах, другие были доступны пользователю.

Эти регистры использовались следующим образом:

восемь регистров были регистрами специального назначения, используемыми для организации процедур;

восемь регистров содержали адреса основных команд, экстракоды и директивы для прерывания, обеспечивающие быстрое переключение процедур;

несколько регистров предназначалось только для системных программ;

несколько регистров могли с некоторыми ограничениями применять пользователи;

несколько регистров использовалось различными компиляторами для хранения индексов и адресов описания данных для адресной модификации.

Однако выяснилось, что, кроме нескольких случаев применения в программах, написанных в машинных кодах, эти регистры использовались весьма неэффективно, особенно в машинных программах, транслированных с языков высокого уровня. Следовательно, в действительности введение этих регистров не давало значительного сокращения числа обращений к памяти и не сглаживало несоответствия в скоростях работы центрального процессора и основной памяти.

Организация памяти машины ATLAS

Иерархическая структура памяти автоматически превращалась с помощью системы в единую одноуровневую память. Пользователю не надо было заботиться о проблемах перераспределения памяти; механизм передачи данных был совершенно неизвестен пользователю. И ферритовая память, и память на магнитных барабанах были разделены на логические страницы объемом 512 слов. Хотя объем памяти в системе составлял только 32 страницы (емкость ферритовой памяти 16К слов), а память на барабанах содержала 96К слов, длина слова для адресации памяти для всех пользователей составляла 20 бит, позволяя адресовать до 10^6 слов. Таким образом, первичная память имела емкость только 32 страницы, тогда как емкость вторичной памяти составляла 2048 страниц.

Так как каждая логическая страница могла размещаться в любой из 32 физических страниц, для того чтобы ускорить процесс формирования адреса, система была снабжена набором из 32 ассоциативных адресных регистров страницы (АРС), по одному адресному регистру для каждой физической страницы; при этом формирование адреса обеспечивалось простым поиском признака.

Старшие 11 разрядов логического адреса служили логическим номером страницы, или меткой, которая сравнивалась ассоциативным образом с содержимым меток всех 32 регистров. При сравнении соответствующий АРС давал декодированный физический номер страницы, который, будучи связан с начальным номером слова, обеспечивал выработку соответствующего физического адреса. Если такого сравнения не было, вызывалась программа прерывания для выполнения

какой-либо передачи данных между ферритовой памятью и памятью на барабанах.

Для того чтобы минимизировать время передачи данных между ферритовой памятью и памятью на барабанах и максимизировать процент удачных запросов на обмен, была создана интеллектуальная *обучающаяся программа* переписи информации с магнитного барабана. Эта обучающаяся программа эквивалентна известным алгоритмам перемещения. Одна дорожка на каждом барабане использовалась для идентификации абсолютного положения блока информации по поверхности барабана. Были сделаны попытки минимизировать время записи передаваемой информации путем записи необходимого блока данных в любое доступное свободное место на любой барабан, а проведение передачи данных отмечалось во вспомогательном указателе. Этот указатель менялся только тогда, когда имела место фактическая передача данных.

В начале программы обмена данными с памятью на барабанах было необходимо найти абсолютное положение требуемой страницы или блока на барабане. Такая программа поиска запускалась, как только получала запрос на чтение информации с барабана. Далее, чтобы ускорить обмен данными между ферритовой памятью и памятью на барабанах, а также совместить время работы обучающейся программы с периодом ожидания чтения данных при их передаче или с периодом самой передачи данных, в ферритовую память была введена пустая, или свободная, страница. В момент определения абсолютного положения требуемого блока на барабане выдавалась команда на передачу данных в эту свободную страницу ферритовой памяти, причем передача данных начиналась автоматически, как только барабан достигал соответствующего углового положения. В адресный регистр этой свободной страницы записывался соответствующий номер блока, переданного с барабана.

Управление процессом передачи данных с барабана в свободную страницу ферритовой памяти составляло половину времени работы обучающейся программы по обмену информацией. Как только завершалась передача данных из барабана в пустую страницу ферритовой памяти, обучающаяся программа настраивалась на выборку соответствующей страницы ферритовой памяти для передачи ее в память на барабанах. Это необходимо, чтобы освободить место для записи при следующем запросе на чтение данных с барабана. Обучающаяся программа ведет выбор блока, удобного для передачи, весьма тщательно, чтобы минимизировать время передачи и максимизировать процент удачных запросов на обмен. Для того чтобы у обучающейся программы была информация,

с которой она может работать, и не осуществлять выбор страницы для передачи на магнитный барабан случайным образом, с каждой страницей был связан разряд «использования». В этом разряде появляется единица при любом обращении к программе. Обучающаяся программа могла считывать эти 32 разряда (со всех АРС), причем при считывании они автоматически обнулялись. Однако частота, с которой опрашивались эти разряды, определялась не реальным временем, а временем выполнения определенного числа команд в основной памяти, а именно: разряды опрашивались каждый раз после выполнения 1024 команд.

Кроме постраничных разрядов «использования» были введены два других динамических параметра, связанных с каждой страницей: 1) общая длительность использования данной страницы (t) и 2) общее время отсутствия обращения к данной странице (T). Частота, с которой опрашиваются разряды «использования», определялась значениями t и T.

Страница, которую следовало «перебросить» с ферритовой памяти на магнитный барабан, выбиралась на основании следующих трех простых тестов (в порядке приоритета): выводится из ферритовой памяти страница, у которой

1) $t > T + 1$;
2) страница с ненулевым t ($t \neq 0$) и максимальным $(T - t)$;
3) максимально значение T при всех $t = 0$.

$t = 0$ показывало, что к данной странице есть обращение.

Согласно первому правилу, выбиралась страница, которая не вызывалась самое длительное время, и, следовательно, наименее вероятно, что она будет использоваться программой. Согласно второму правилу, не рассматривались страницы, которые использовались в текущий момент, а выбиралась страница, которая дольше всех не вызывалась программой. Если таких страниц не находилось, выбиралась страница из числа опрашиваемых, но имеющая максимальное T (общее время отсутствия обращения к данной странице). Если же к этой странице было новое обращение, то T устанавливалось в нуль и считалось, что такая ошибка более не повторится.

Хотя машина ATLAS не была полностью мультипрограммной, она продемонстрировала возможность существования простого и изящного одноуровневого автоматизированного механизма памяти. Несмотря на то что в памяти могло храниться множество программ, процессор не переключался на обработку другой программы на время поиска адреса передачи. Процессор в основном простаивал во время смены программ и, таким образом, серьезно ограничивал производительность системы. Хотя было известно, что опытный программист, детально знакомый с внутренней структурой машины,

может организовать эффективный процесс передачи данных, тем не менее это было чрезвычайно сложно для пользователя, и в результате наблюдалась потеря производительности. Опыт использования машины ATLAS показал, что систему следует проектировать таким образом, чтобы снабжать пользователей механизмом переключения процессора на работу с другим пользователем всякий раз, когда процессор простаивает при выполнении системой длинных операций (особенно во время работы устройств ввода-вывода и передачи страниц данных). Было также выяснено, что вместо деления памяти на блоки фиксированной длины (страницы) должна быть предусмотрена возможность изменения их длины (т. е. блоки следует делить на сегменты).

Архитектура ЭВМ ATLAS оказала огромное влияние на машины второго и третьего поколений шестидесятых годов. До появления машины ATLAS решения проблем, вызываемых иерархией памяти, были либо весьма дорогими, либо неэффективными. Проблемы иерархии памяти решались использованием либо очень большой и дорогой ферритовой памяти, как, например, в машине MIT [P7], имеющей память емкостью 2,5 Мбит, либо управляемой пользователем памяти с двухуровневой иерархией, т. е. небольшой памяти на ферритовых сердечниках емкостью 40 000 бит, расширяемой до 640 000 бит с помощью памяти на магнитных барабанах, как в ЭВМ Ferranti Mercury. В последней системе пользователь должен был сам заботиться об организации всех передач данных.

Несмотря на то что машина ATLAS была спроектирована около двух десятилетий назад, впечатляет тот факт, что разработчики предложили остроумную и дешевую схему реализации системы виртуальной памяти. Хотя это и не понимали в то время, когда машина проектировалась, она имела черты микропрограммной страничной организации, присущие современной виртуальной памяти с кэшем. Преимущества введения предлагаемых процедур были тщательно обоснованы и представлено очень элегантное решение задачи построения виртуальной памяти для решения проблемы управления памятью.

Учитывая опыт проектирования машины ATLAS, Килбурн и другие предложили использовать память на туннельных диодах в сочетании с быстродействующей ферритовой памятью или даже быстро и медленно действующие устройства ферритовой памяти для построения систем высокой производительности. Такой подход имел прямое влияние на систему ETL-MK6.

В ЭВМ ETL-MK6 [T1] также применялась трехуровневая иерархия памяти, для которой характерно особое использование нескольких быстродействующих ЗУ. На самом низком

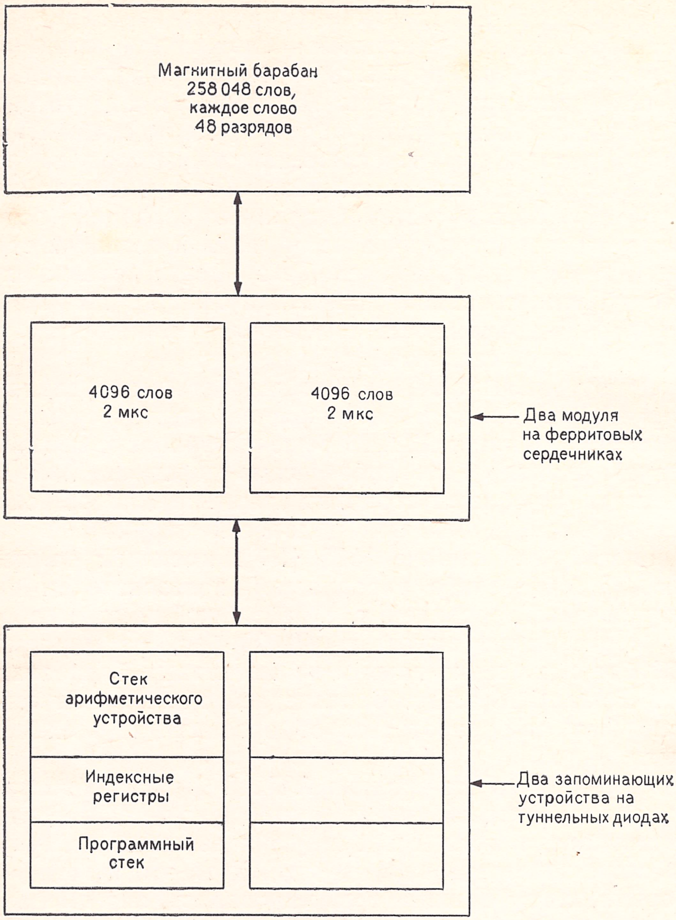


Рис. 7.3. Трехуровневая иерархия памяти системы ETL-MK6.

уровне была память на магнитных барабанах емкостью 258 048 слов по 48 бит каждое. На промежуточном уровне находились две ферритовые памяти емкостью по 4096 слов с временем выборки 2 мкс, а на высшем уровне иерархии были две памяти на туннельных диодах емкостью по 64 слова каждая и с временем выборки 250 нс (рис. 7.3). Память на туннельных диодах в свою очередь делилась на три части: программный стек емкостью 32 слова, арифметический стек емкостью 64 слова и 32 индексных регистра. Арифметический стек и индексные регистры были доступны программистам, а программный стек был недоступен. Хотя с позиции разработчика ЭВМ ETL-MK6 имела три уровня иерархии памяти,

для пользователя это была одноуровневая память емкостью 2^{18} слов. 32 слова из программного стека выбирались чрезвычайно быстро, и стек активно использовался для хранения выполняемых текущих 64 шагов программы и команд после выборки их из памяти. Только за счет запоминания нескольких команд и активного использования циклического режима работы программ оказалось возможным достигнуть значительного увеличения производительности при выполнении программ.

Память с просмотром соседних ячеек

Для решения проблемы согласования значительной разницы в скорости работы логических устройств и устройств памяти Блюм, Коэн и Портер [B1] предложили структуру памяти с просмотром соседних ячеек вместо структуры с предварительным просмотром ячеек. Основываясь на принципе локального наблюдения за поведением выполняемых программ, они предложили использовать небольшую быстродействующую ассоциативную память, состоящую из набора быстродействующих регистров, полностью недоступных программистам (т. е. программисты не могли непосредственно обращаться к этой памяти), хотя в принципе эти регистры являлись частью основной памяти. Эта ассоциативная регистровая память служила главным образом ускорителем передачи информации. Как показано на рис. 7.4, каждый ее регистр состоял из трех частей: 1) адреса памяти или ячейки, 2) данных или содержимого ячейки и 3) индикаторов использования (управляющая часть).

Части этих регистров, содержащие адреса памяти, были связаны с логической схемой сравнения, которая могла сравнивать одновременно адреса памяти, находящиеся в регистре, с адресом памяти, затребованным процессором. Если имело место соответствие, то содержимое ячейки сразу же становилось доступным процессору, и тем самым устранялась необходимость выборки информации из памяти. Только тогда, когда данные, запрашиваемые процессором, отсутствовали в ассоциативной регистровой памяти, включался механизм передачи данных из основной памяти и содержимое соответствующей ячейки вместе с ее адресом помещалось в соответствующие места ассоциативной регистровой памяти.

Чтобы эффективно выполнить алгоритм перемещения и определить, какой из регистров ассоциативной регистровой памяти поместить в основную память, Блюм и др. отошли от классического цифрового подхода и применили аналоговые индикаторы использования, сопоставив каждому регистру конденсатор. Конденсатор подзаряжался каждый раз, когда

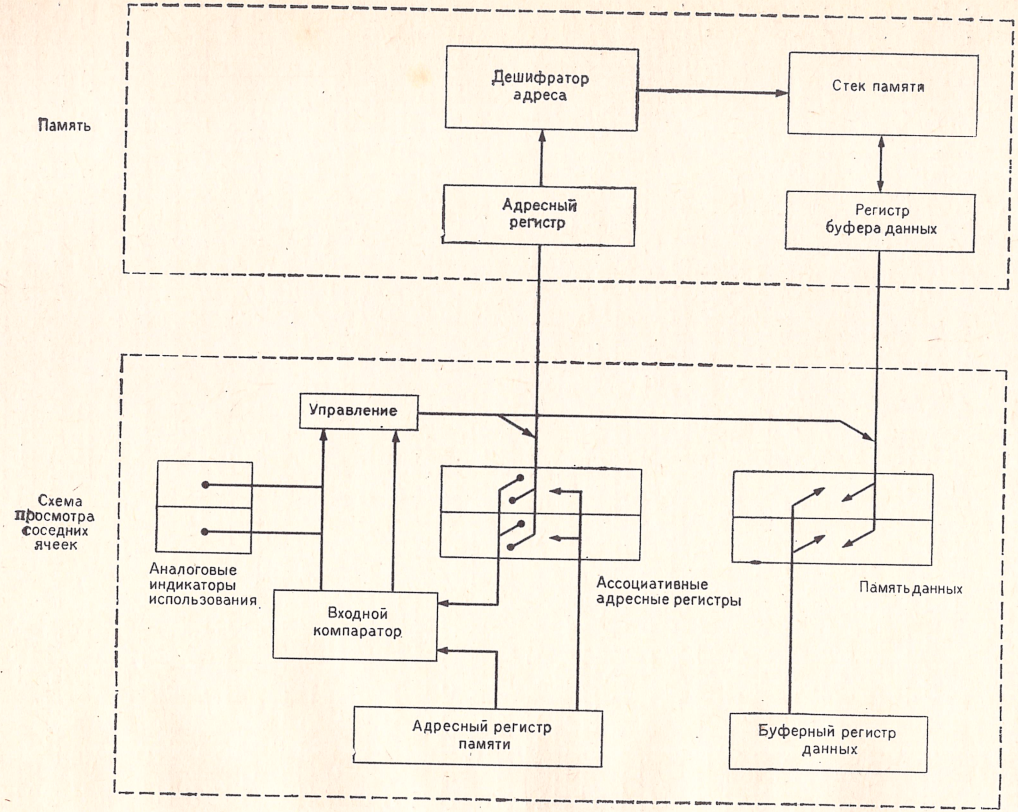


Рис. 7.4. Память с просмотром соседних ячеек Блюма, Коэна и Портера.

выбирался соответствующий регистр этой памяти. Таким образом, в любой момент заряд, остающийся на конденсаторе, определял регистр, к которому обращались в последний момент. Конденсатор, имеющий наименьший заряд, указывал на регистр, который не выбирался дольше всех остальных. Авторы предложили также более сложный подход с использованием двух конденсаторов для каждого регистра, один из которых заряжался до полного напряжения, а другой подзаряжался с постоянным инкрементом каждый раз при выборке соответствующего регистра.

Используя очень сложную структуру управляющих и индексных регистров, они смогли значительно сократить число команд в цикле, уменьшив таким образом число регистров в ассоциативной регистровой памяти. Они также пришли к выводу, что число ассоциативных регистров для обеспечения эффективной работы должно быть по крайней мере равно числу команд в цикле плюс число аккумуляторов и индексных регистров, используемых в цикле работы программы.

Ли [L1] предложил улучшенную версию такой ассоциативной регистровой памяти. Он установил, что выигрыш в производительности такой системы памяти зависит от четырех факторов: 1) емкости быстродействующей ассоциативной памяти, 2) отношения скорости работы ассоциативной памяти к скорости работы основной памяти, 3) алгоритма переписи и 4) объектной программы, выполняемой в этот момент системой. Для увеличения производительности он предложил слегка видоизмененную схему. Также как ассоциативная регистровая память Блюма и др., память Ли состояла из трех частей: адресной части, данных и управляющей части. Но управляющая часть была организована несколько иначе. У Ли управляющая часть состояла из одного разряда, который указывал, изменялись ли данные с момента их последней передачи из основной памяти, и второго разряда, который использовался в качестве указателя. Ячейка, имеющая единицу в разряде указателя, являлась кандидатом на замену или очистку при передаче нового слова из основной памяти. Из всех слов, находящихся в ассоциативной памяти, только одно могло иметь единицу в разряде указателя, остальные указатели были установлены в нуль.

Вспомогательная память Уилкса

М. В. Уилкс [W1] в 1965 г. рассмотрел использование быстродействующей ферритовой памяти емкостью 32К слов, служащей *вспомогательной* памятью для бо́льшей и медленнее действующей ферритовой памяти емкостью 1 млн. слов, в качестве средства для выполнения динамического распре-

деления памяти и получения эффективного времени выборки, равного циклу работы быстродействующей ферритовой памяти. Под вспомогательной памятью он подразумевал структуру, которая позволяла накапливать в быстродействующей памяти необходимую информацию автоматически. Информация, поступавшая из медленнодействующей памяти, становилась доступной процессору для последующего использования; это позволяло избежать неприятностей, связанных с длительным временем выборки из основной памяти. Уилкс создал основу для удовлетворительной двухуровневой системы памяти, не требующей значительного усложнения оборудования. Он рассмотрел несколько структур вычислительной системы, которая имела память емкостью 32К слов с временем выборки 1 мкс, поддерживаемую большой ферритовой памятью емкостью 1 млн. слов с временем выборки 8 мкс.

В простейшей структуре Уилкс предложил разделить программы на блоки емкостью 32К слов с тем, чтобы каждый пользователь имел один или несколько блоков для своих программ. Был введен базовый регистр для большой ферритовой памяти, содержащий начальный адрес работающего в данный момент блока вспомогательной памяти. Каждому базовому регистру было сопоставлено два разряда признака; первый разряд показывал, перенесен ли блок (слово) в быстродействующую память, а второй разряд показывал, менялось ли содержимое скопированного блока. Вспомогательная память управлялась в зависимости от локального поведения программы; использовался флаговый алгоритм свопинга для обмена информацией между быстрым и медленным уровнями памяти, чтобы уменьшить число передач и улучшить эффективность работы системы. Уилкс также предлагал избегать передачи всех 32К слов данных из основной во вспомогательную память, так как только небольшая часть из этих 32К слов использовалась в рабочей программе.

При любом обращении к памяти сначала выбиралась быстродействующая память, и если первый разряд регистра признака был установлен в единицу, то медленнодействующая память не опрашивалась. Если слово в быстродействующей памяти изменялось, то второй разряд регистра признака устанавливался в единицу (рис. 7.5). Таким образом, постепенно быстродействующая память накапливала все слова, используемые в работе данной программы. Когда запускалась новая программа, число в базовом регистре изменялось в соответствии с номером используемого блока емкостью 32К слов, а для перемещения информации из вспомогательной памяти в медленнодействующую применялся алгоритм разделителя файлов (FS-алгоритм). Регистры, имеющие код 00 в первых разрядах признаков, не затрагивались, также как

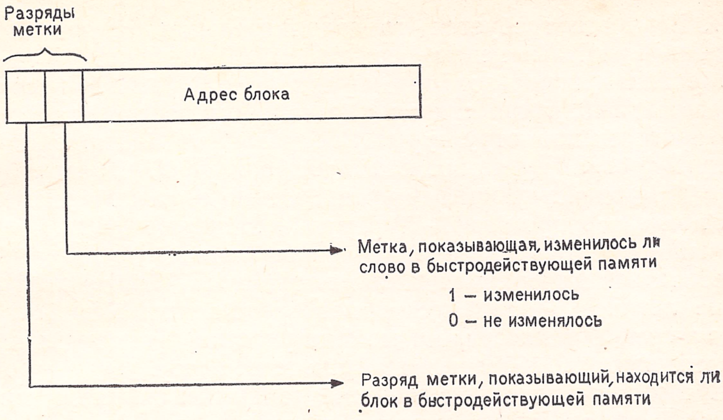


Рис. 7.5. Схема Уилкса с разрядами меток.

и регистры с кодом 10 в разрядах признаков; затрагивались только регистры, имеющие код 11 в разрядах признаков. Так как эта схема работы не использовала алгоритм сквозной записи, то возникала необходимость переписи измененного блока из быстродействующей памяти в медленно действующую.

Уилкс предложил также несколько схем организации вспомогательной памяти для обслуживания разных блоков основной памяти. Бо́льшая часть концепций, предложенных им, используется в современных системах с кэш-памятью, например:

1) разряды достоверности ЭВМ IBM 360/85, сопровождающие блоки информации, есть не что иное, как динамические разряды признаков, предложенные Уилксом;

2) передача по возможности не всего буфера, а его части, что используется в системе IBM 360/85.

Система Андерсона и Глейзера

Андерсон и Глейзер [A1] рассмотрели вопросы, связанные с использованием иерархической системы памяти в сверхбыстродействующих вычислительных системах. Иерархия, исследуемая ими, состояла из следующих уровней:

1. Относительно небольшая сверхбыстродействующая память с фиксированным временем доступа, сравнимая по быстродействию с самыми современными логическими схемами.

2. Вторичная память с фиксированным временем выборки намного бо́льшего объема и меньшего быстродействия.

3. Еще более медленная, третья по уровню память с фиксированно-переменным временем доступа, такая, как память на дисках, магнитных барабанах или фотопластинках,

4. Если это требуется, то дополнительная память, включающая магнитные ленты, перфокарты и т. п.

Механизм использования иерархической памяти, рассматриваемый ими, тесно связан с принципом построения основной программной структуры. Рассматривая блочную структурную программу языка Алгол 60, они указали, что объем сверхбыстродействующей памяти должен допускать размещение элементов программы (команд и данных), «активных» в данный момент, тогда как бóльшая память могла бы использоваться для размещения тех элементов программы, которые логически находятся в непосредственной близости от выполняемого в настоящий момент участка программы и которые могут потребоваться программе не сразу.

В соответствии с их экспериментальными оценками минимальный объем сверхбыстродействующей памяти лежит в пределах от 128 до 256 слов, причем разумным пределом считалась емкость от 512 до 1024 слов; они сделали вывод, что емкость от 1024 до 2046 слов должна быть достаточной, чтобы удовлетворить запросы выполняемых в данный момент элементов программы для большинства программ.

В то время как Уилкс выступал за концепцию вспомогательной памяти только для операндов, Скэротт [S1] также в 1965 г. расширил этот подход не только для операндов, но и для команд. Скэротт рассматривал организацию вспомогательной памяти с трех основных точек зрения: 1) характера адресной последовательности (команд или данных); 2) характера операций над данными (модифицированные или нет) и 3) необходимости соответствия иерархии памяти данному программному обеспечению.

Вместо введения единой вспомогательной памяти Скэротт предлагал использовать разные вспомогательные памяти для операндов и команд. Также основываясь на характере адресной последовательности, он считал, что емкости этих двух памятей должны быть различны. Буфер команд вмещал 16 слов, а буфер данных — 64 слова. Оба буфера поддерживались вспомогательной памятью емкостью 16К слов. Чтобы еще улучшить механизм передачи данных, Скэротт предложил, чтобы буфер команд представлял собой память только для считывания, тогда как буфер операндов мог выполнять операции чтения и записи информации.

Ввиду того что команды не модифицировались процессором, не было необходимости возвращать какую-либо информацию из буфера команд в основную память. В то же время операнды могли модифицироваться процессором; поэтому он должен был быть организован, как буфер, с чтением и записью. Чтобы улучшить передачу данных, Скэротт ввел модифицируемый разряд для каждой ячейки буфера, который

устанавливался только в том случае, если данные, относящиеся к данной конкретной ячейке буфера, изменялись выполняемой программой. Позднее этот модифицируемый разряд указывал, требуется ли данное слово вывести обратно в основную память или нет. Схема с модифицируемым разрядом подобна схеме, предложенной Ли и известной теперь также как стратегия с *«обратной»* записью. В случае отсутствия ячейки в буфере (при попытке записи процессором операнда в ячейку, не найденную в буфере операндов) модифицировалась только буферная память операндов. Если при этом заменялось слово в быстродействующей памяти, то соответствующее слово должно было быть выбрано. Если в этом слове модифицируемый разряд был установлен, то это слово передавалось в основную память.

Скэротт утверждал также, что существует сильная корреляция между соответствующим программным обеспечением, структурой иерархии памяти и емкостью буфера. Он указал, что, в то время как небольшая емкость буфера может быть достаточной для научных задач, для коммерческих задач может потребоваться бо́льшая емкость буферной памяти. Идеи Скэротта о модифицируемом разряде, неоднородном объеме буферной памяти и ее различных размерах оказали большое влияние на современные ЭВМ.

В 1967 г. Гибсон [G1] впервые провел подробное исследование полезности блочного обмена информацией с большим количеством слов между памятью второго уровня иерархии и локальной (буферной) памятью. Он интуитивно понял, что путем передачи блоков слов вместо одного слова за одно обращение и использования локального характера поведения программ можно уменьшить первоначальное время выборки благодаря выполнению операции передачи данных, преобразования адресов и т. п. сразу над большим количеством слов. Среди фундаментальных вопросов, на которые он хотел получить ответ, были следующие:

Каким должен быть оптимальный размер блока?

Какова должна быть емкость локального буфера у процессора?

Какой алгоритм перемещения следует применять?

Как влияет исполняемая программа на эффективность передачи информации блоками?

Чтобы ответить на эти вопросы, он отобрал длинные последовательности по 7000 команд из двадцати программ для ЭВМ. Каждая программа осуществляла приблизительно три миллиона адресных пересылок, которые для анализа были разделены на 200 000 последовательностей. При моделировании Гибсон варьировал размеры блоков от четырех до 248 слов в блоке, а размер локального буфера — от 32 до

8192 слов. Он проверил также 15 различных алгоритмов перемещения. Ниже приведено несколько важных выводов, к которым он пришел.

1. Размер блока от 4 до 16 слов наиболее эффективен с точки зрения использования емкости локального буфера, причем, чем короче блок, тем лучше. Полезность экстракодов в блоке, казалось, уменьшается с увеличением длины блока. Он обнаружил также, что, в то время как число обращений вне локальной памяти увеличивается с уменьшением длины блоков, число слов, передаваемых в локальную память, уменьшается с уменьшением длины блоков.

2. При локальном буфере, имеющем емкость памяти от 2 до 4К слов, для большинства программ процент удачных обращений к памяти близок к 100.

3. Хотя некоторые алгоритмы перемещения с ассоциативным отображением давали лучшие результаты, чем алгоритм конгруэнтного отображения, вообще влиянием алгоритма перемещения на производительность системы можно пренебречь.

4. Для достижения высокой производительности системы структура адресации важнее, чем длина программы.

5. Меньшие размеры блоков давали бóльшую плотность передачи информации между процессором и его локальной памятью, и чрезвычайно важно выбирать длину блока в соответствии с емкостью локальной памяти, чтобы достичь желаемой скорости передачи информации.

Разделив потоки информации на потоки команд и потоки данных, Сиссон [S6] провела дальнейшее моделирование и оспорила результаты Гибсона о том, что меньшая длина блока дает более высокую производительность. Она обнаружила также, что, возможно, из-за локального поведения программы выбор команды происходит с большей скоростью, чем выбор операнда, и рекомендовала использовать различные емкости буферов для команд и данных.

В 1969 г. Ли [L4] официально опубликовал результат своего изучения методом моделирования схемы ассоциативной регистровой памяти емкостью от 4 до 256 слов с целью оценки улучшения производительности при использовании ее в ЭВМ PDP-1. Ли пользовался тремя простыми объектными программами: 1) программой извлечения корней (это в основном вычислительная программа, вызывающая подпрограммы с плавающей запятой, и программы десятично-двоичного преобразования); 2) программой MACRO (макроассемблер ЭВМ PDP-1 для трансляции коротких программ с магнитной ленты); 3) программой моделирования ассоциативной регистровой памяти (копия модели). Ли обнаружил, что при отношении скоростей работы буферной и основной памяти,

равном 10, и в предположении, что время выборки основной памяти составляет 1 мкс, буферной памяти — 100 нс, емкость ассоциативной буферной памяти равна 128 ячейкам, можно получить эффективную скорость работы, только в два или три раза превышающую скорость выборки из буферной памяти, т. е. достичь время цикла выборки от 250 до 400 нс.

Затем Ли исследовал два алгоритма очистки (перемещения): алгоритм побитовой обработки и кольцевой алгоритм. Он утверждал, что сложный алгоритм перемещения невыгоден с точки зрения стоимости. Действительно, еще в 1964 г. Ли [L2] высказал мысль о большем удобстве передачи информации от памяти второго уровня иерархии к локальному буферу процессора небольшими блоками, чем отдельными словами, или, как это сделано в ЭВМ ATLAS, длинными страницами (512 слов). Такая схема была формально исследована Гибсоном.

7.2. БОЛЬШИЕ СИСТЕМЫ С КЭШ-ПАМЯТЬЮ. СИСТЕМЫ С КЭШ-ПАМЯТЬЮ ФИРМЫ IBM

Хотя концепция иерархии памяти и использования быстродействующей буферной памяти для получения бо́льшей производительности известна уже давно, кэш-память была впервые использована в ЭВМ IBM 360/85 — большой универсальной вычислительной системе коммерческого назначения [L3]. В ЭВМ IBM 360 модели 85 иерархическая память, состоящая из относительно небольшой, но быстродействующей полупроводниковой памяти (емкость от 16 до 32 Кбайт, время выборки 80 нс) и большой относительно медленнодействующей основной памяти (емкость от 512 до 4096 Кбайт, время выборки 1,04 мкс), использовалась для достижения большой производительности и высокой пропускной способности. Небольшая быстродействующая полупроводниковая память объединена с центральным процессором и называлась кэш-памятью. Эта память недоступна пользователю, т. е. программист к ней адресоваться не мог (термин «кэш» означает спрятанный). Кэш-память содержала информацию, наиболее часто запрашиваемую процессором, в результате чего для процессора уменьшалась необходимость получать данные из основной памяти. Таким образом, бо́льшая часть памяти для процессора могла рассматриваться как быстродействующая, и только в случае отсутствия необходимых процессору данных в кэш-памяти вызывалась основная память и в процессор вводились новые данные.

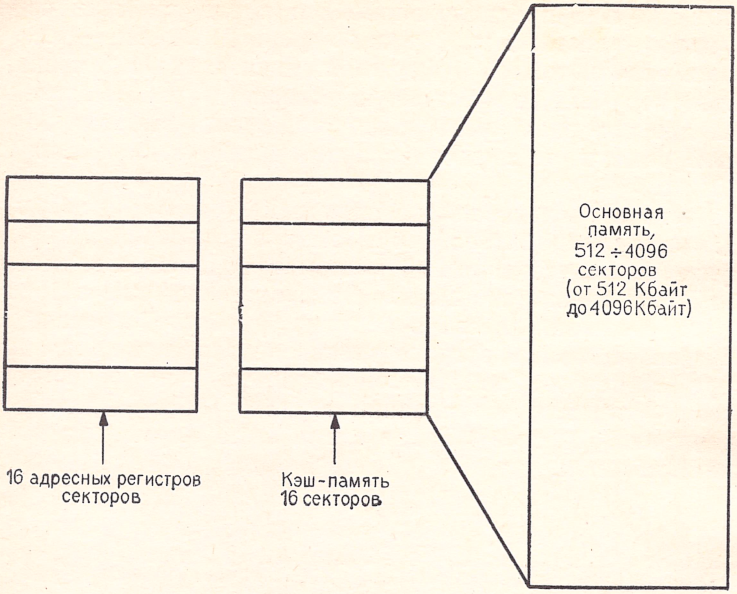


Рис. 7.6. Кэш-память системы IBM 360/85.
Каждый сектор = 1 Кбайт, один сектор = 16 блоков, один блок = 64 байт.

Организация кэш-памяти

В модели 85 ЭВМ IBM 360 и основная, и кэш-память разделены на логические секторы, или страницы фиксированной длины. Каждый сектор состоял из одного килобайта последовательной информации. Кэш-память включала 16 секторов (16 Кбайт), а основная память содержала от 512 до 4096 секторов (от 512 до 4096 Кбайт). Так как в кэш-памяти размещалось только 16 секторов, в любой заданный момент времени лишь 16 секторов с информацией из основной памяти могли быть в кэш-памяти, и только 16 секторов основной памяти имели соответствующие им секторы в кэш-памяти. Поэтому во время выполнения программы должно быть установлено какое-либо соответствие между секторами основной и кэш-памяти. Это достигалось введением 16 адресных регистров секторов, которые служили метками памяти для секторов основной памяти и содержали адреса секторов основной памяти, которым соответствовали секторы в кэш-памяти (рис. 7.6).

Соответствие между секторами кэш-памяти и секторами основной памяти устанавливалось динамически, а перемещение выполнялось с помощью алгоритма LRU. Были сделаны попытки связать секторы кэш-памяти с теми секторами основ-

ной памяти, которые используются программой. Был введен список динамически распределяемых приоритетов активностей секторов, причем сектор, к которому процессор обращался в последнюю очередь, помещался наверху списка (что указывало на его наивысший приоритет), а сектор, к которому процессор обращался в первую очередь, устанавливался внизу этого списка (что указывало на его наинизший приоритет).

По мере того как кэш-память удовлетворяла запросы процессора, в ней просто передвигались указатели приоритетов, а сами данные не перемещались. Однако если процессору требовалась информация из сектора, который не отображен в кэш-памяти (например, это было обращение к неиспользовавшемуся ранее сектору основной памяти), один из секторов кэш-памяти необходимо было вывести в основную память, чтобы освободить место в кэше. Выбираемый для этого сектор кэш-памяти находился внизу списка приоритетов (к нему не обращались дольше всего и поэтому, вероятнее всего, не будут обращаться и в будущем).

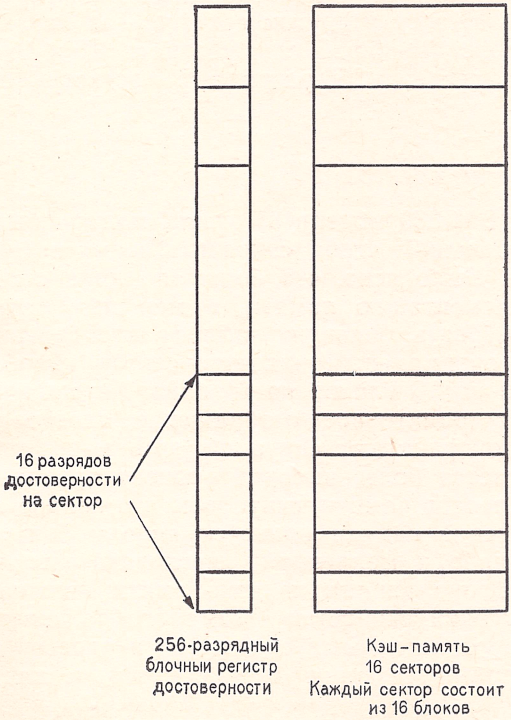


Рис. 7.7 Кэш-память системы IBM 360/85 и 256-разрядный регистр блока достоверности.

Обращение к конкретному сектору перемещало его в начало списка приоритетов; при этом все предыдущие приоритеты сдвигались на одну позицию вниз. Секторы, лежащие ниже, при этом не затрагивались. Отметим, что все это не означало фактического перемещения секторов внутри кэш-памяти. Просто происходило перемещение указателей в соответствии с новыми приоритетами.

Подготовка сектора к передаче его содержимого из кэш-памяти в основную память заключалась просто в установке соответствующей метки в кэш-памяти. Более того, в секторах не происходило перемещения данных. Для минимизации времени перемещения каждый сектор был организован в виде 16 блоков; каждый блок имел длину 64 байт, и блоки загружались в кэш-память по запросам.

Так как каждый сектор состоял из 16 блоков, в каждый блок вводилась некоторая специальная информация, чтобы идентифицировать те блоки в кэш-памяти, которые относятся к отображаемому в этот момент сектору. В каждом блоке устанавливался бит опознавания, когда данный блок загружался, и этот бит сбрасывался, когда данный сектор кэш-памяти отображался в другом секторе. Так как кэш состоял из 16 секторов (в общей сложности из $16 \times 16 = 256$ блоков), был введен 256-разрядный регистр опознавания (рис. 7.7).

Преобразование адресов

В ЭВМ IBM 360 модели 85 длина сектора была выбрана достаточно большой, чтобы сократить время на преобразование адресов. Было ясно, что большая длина сектора может вызвать фрагментацию памяти, а меньшая длина сектора может потребовать бо́льшую таблицу преобразований. Считалось, что оптимальным является сектор длиной 1 Кбайт, разделенный на 16 блоков по 64 байт. Хотя было ясно, что полностью ассоциативное отображение адресов минимизирует число конфликтных ситуаций при выборке, благодаря чему достигается более высокий процент принятых к обслуживанию запросов памяти, считалось, что полное ассоциативное отображение памяти приведет либо к большому числу сравнений, либо к медленному последовательному поиску. Так как выборка данных из кэш-памяти должна осуществляться с высокой скоростью, схема отображения сектора с полностью ассоциативными сравнениями адресов с 16 адресными регистрами секторов была принята в качестве оптимальной.

Полный адрес в ЭВМ IBM 360/85 состоял из трех компонент: адреса сектора, адреса блока и адреса байта (рис. 7.8).

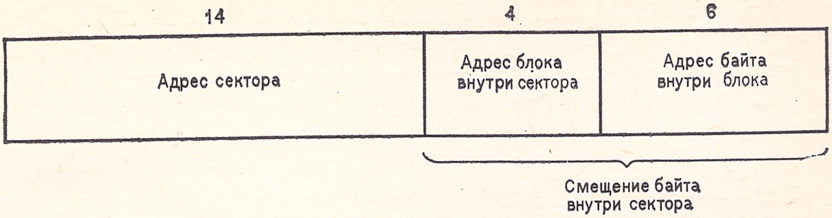


Рис. 7.8. Логическая адресация системы IBM 360/85.

Каждый сектор имел 16 блоков, каждый блок включал 64 байт. Так как каждый сектор содержал 1К следующих друг за другом байтов, то блоки отображались последовательно и расположение любого байта в конкретном секторе было фиксировано. Младшие шесть разрядов адреса определяли адрес байта внутри блока, следующие четыре разряда — адрес блока внутри сектора, а старшие 14 разрядов — конкретный адрес сектора как в основной памяти, так и в кэш-памяти. Итак, младшие 10 разрядов фактически определяли смещение байта внутри конкретного сектора.

Так как в кэш-памяти находилось только 16 секторов, то 14 старших разрядов адреса необходимо было сравнивать с метками, хранящимися в кэше, чтобы определить, отображен ли соответствующий сектор основной памяти в кэш-памяти и, если отображен, то в каком месте. Если имело место соответствие любого запрашиваемого адреса с меткой, то декодированное содержимое выбранной ячейки определяло старшие четыре разряда адреса кэш-памяти. В таком случае эти четыре разряда объединялись с четырьмя разрядами адреса блока (при формировании логического адреса), формируя соответствующий восьмиразрядный адрес блока кэш-памяти, после чего устанавливался соответствующий бит опознавания. Если соответствующий бит опознавания установлен в единицу (т. е. блок опознан), то нужный блок уже размещен в кэш-памяти (рис. 7.9).

Младшие шесть разрядов определяли адрес байта блока, запрашиваемого процессором. Однако, если блок не опознан (бит опознавания равен нулю), его требовалось загрузить в кэш-память. Если запрашиваемый адрес сектора не соответствовал никакому адресу сектора в кэш-памяти, формировался признак отсутствия в кэше адреса сектора, и тогда следовало снова адресоваться к гому же сектору.

Чтобы избежать необходимости обратной переписи всего содержимого сектора за время, отведенное на перезагрузку, в кэш-памяти ЭВМ IBM 360/85 использовался метод сквозной записи. Этот метод заключается в том, что все операции

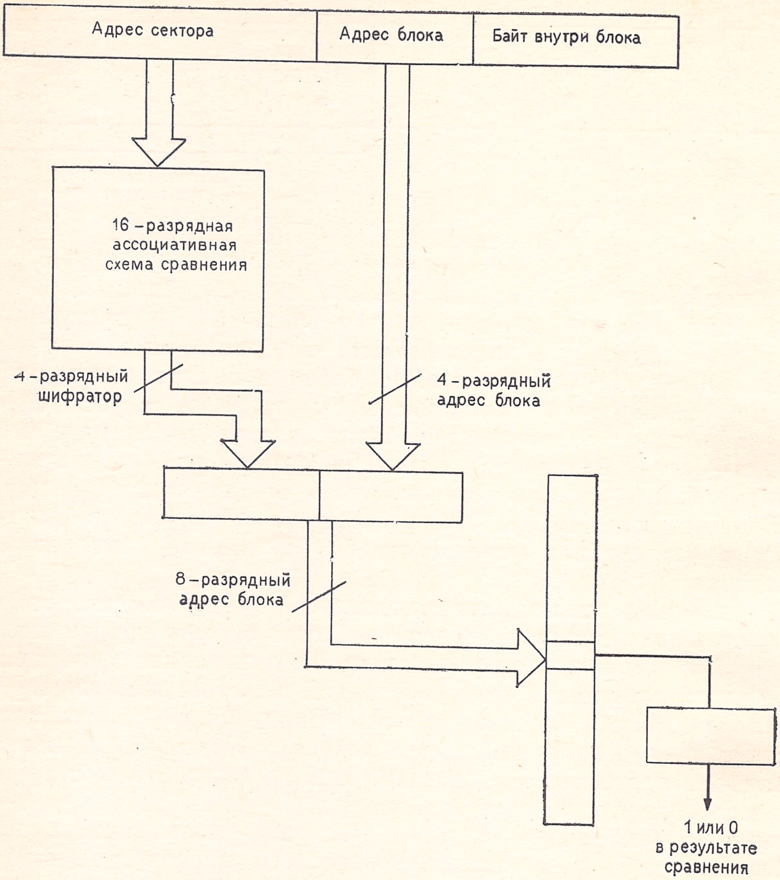


Рис. 7.9. Механизм преобразования адресов системы IBM 360/85.

записи информации из процессора изменяют данные в ячейках соответствующего сектора основной памяти, даже если этот сектор основной памяти соответствует содержимому в кэш-памяти. Если сектор основной памяти имеет соответствующий ему сектор в кэш-памяти, то информация в последнем также обновляется. Если в кэш-памяти такого сектора нет, то кэш-память не вызывается совсем (т. е. ни бит опознавания, ни список приоритетов не изменяются). Так как в основной памяти всегда содержатся обновленные данные, то отсутствует проблема устаревания данных и нет необходимости передавать какие-либо данные из кэш-памяти в основную память. Только операция чтения может переобозначить сектор в кэш-памяти, перезагрузить блоки или изменить спи-

сок приоритетов. Таким образом, при введении сквозной записи для всех операций записи операции по переобозначению стека, загрузке блоков и изменению списка приоритетов сводятся к минимуму.

Обычно процессор получает данные из кэш-памяти за два цикла. В течение первого цикла осуществляются поиск меток и контроль достоверности, а во втором цикле выбираются данные. Были приняты меры для обеспечения перекрытия во времени запросов таким образом, чтобы запросы к памяти могли удовлетворяться каждый цикл. Если данных в кэш-памяти не было (для операций чтения), то требовались добавочные циклы, чтобы переписать данные из основной памяти в кэш-память.

Базовая длина слова в памяти для ЭВМ IBM 360/85 равнялась 16 байт. Поэтому два 8-байтовых модуля ферритовой памяти были спарены вместе и работали одновременно. Кроме того, использовалось четырехкратное расслоение для формирования слова в 64 байт, т. е. всего блока. Чтобы еще более увеличить производительность, первым основным модулем памяти, к которому обращались в первую очередь (во время каждой загрузки блока), был модуль, содержащий 16 байт данных, запрашиваемых процессором. Этот модуль запускался первым, и данные из него — кроме того, что они загружались в кэш-память, — направлялись непосредственно в процессор, так что выполнение операции могло начаться немедленно.

Оценка производительности

Чтобы оценить эффективность предложенной кэш-памяти, Липтей [L3] сравнил ее производительность с идеализированной системой 360/85, состоящей из интегральной одноуровневой памяти, работающей со скоростью кэш-памяти, что представляет собой верхний предел производительности модели 85. Было взято 19 магнитных лент (содержащих ~ 250 тыс. команд, выполняемых либо рабочими программами, либо операционными системами), данные которых послужили входной информацией для моделирующей программы с целью сравнения реальной системы 360/85 с идеализированной. Было обнаружено, что реальная система достигает скорости выполнения операций от 66 до 94 % (в среднем 81 %) от идеализированной. Кроме того, было найдено, что средняя вероятность нахождения данных, требуемых процессору, в кэш-памяти достигает неправдоподобно большого значения — 96,8 %. Это объясняется тем, что выбранные программы обладали значительной «локальностью» поведения.

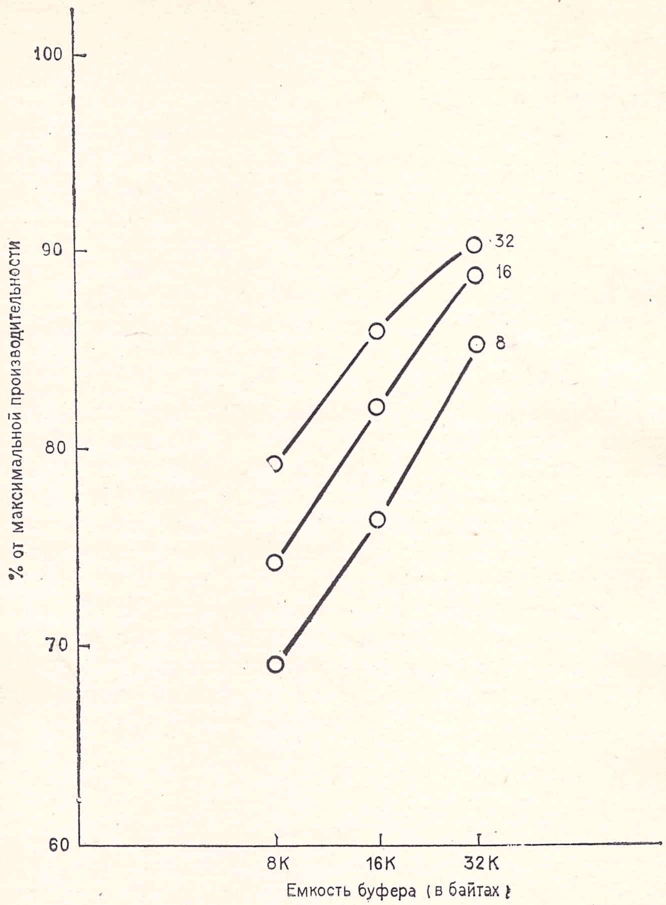


Рис. 7.10. Кэш-память данных системы IBM 360/85. (Длина блока = = 64 байт.)

Было отмечено, что при формировании программных адресов случайным образом вероятность нахождения требуемых данных в кэш-памяти была бы намного меньше одного процента.

Параметры для проектирования кэш-памяти (длина сектора, длина блока, алгоритм перемещения и т. д.) были определены после интенсивного анализа с помощью моделирования. На рис. 7.10 и 7.11 показана производительность кэш-памятей различной емкости с различным числом секторов и различными длинами блока по отношению к идеализированной интегральной одноуровневой системе памяти. На основа-

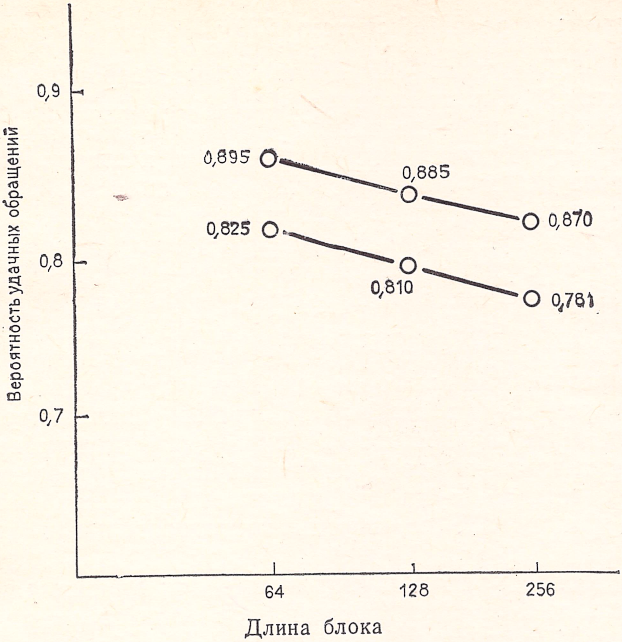


Рис. 7.11. Вероятность удачного обращения к данным в IBM 360/85. (Емкость кэша = 16 секторов.)

нии этих данных объем кэш-памяти был выбран равным 16 Кбайт, длина каждого сектора составляла 1 Кбайт, каждый сектор был разделен на 16 блоков по 64 байт в каждом. Было также исследовано влияние различных алгоритмов перемещения — от алгоритма полного ассоциативного отображения информации до алгоритма прямого отображения. В конце концов все параметры были выбраны так, чтобы получить наилучшее соотношение между ценой и производительностью.

В системе 360/85 была впервые реализована и получила признание концепция применения в больших коммерческих системах кэш-памяти как иерархической системы памяти, имеющей два уровня быстродействия: наносекундный и микросекундный. С тех пор концепция кэш-памяти начала применяться в больших вычислительных системах почти повсеместно. Кэш-память была использована в системе 360 модели 195 и включена также в машины серии 370, модели 370/155 и 370/165. Эти системы описаны в работе [К6].

Кэш-память модели 155 системы 370

ЭВМ 370 модели 155 является высокопроизводительной 32-разрядной вычислительной системой, спроектированной для коммерческих и научных приложений, а также для работы в реальном масштабе времени. Она имеет длительность машинного цикла 115 нс и цикл выборки основной памяти 2070 нс (для выборки 16 байт из памяти). Различие в быстродействии процессора и основной памяти (скорости различаются в 18 раз) сглаживается использованием быстродействующей кэш-памяти. Кэш-память для модели 155 имеет емкость восемь Кбайт и цикл выборки 115 нс (время, равное циклу работы процессора). Кэш-память спроектирована таким образом, что выборка информации может инициироваться процессором каждый машинный цикл. Разрядность шины данных процессора и шины связи между процессором и кэш-памятью равна четырем байтам. Процессор может выбрать четыре байта из кэш-памяти за два машинных цикла, или за 230 нс, восемь последовательных байтов — за 345 нс.

И основная память, и кэш-память для данных разделены на блоки по 32 байт. Каждый блок делится на два полублока по 16 байт. Полублок емкостью 16 байт (счетверенное слово) является основной единицей обмена информацией между основной памятью и кэш-памятью.

Организация кэш-памяти

В ЭВМ системы 370 исполнительный адрес памяти формируется путем сложения содержимого базового регистра, индексного регистра и смещения. Так как модель 155 может работать с физической памятью емкостью до двух мегабайт, то используется только 21 младший разряд физического адреса (разряды с 11 до 31), как показано на рис. 7.12. Физический адрес состоит из трех полей: разряды 28—31 определяют местонахождение внутри полублока, 27-й разряд определяет выбираемый полублок, а разряды 11—26 определяют адрес блока. Логически адрес блока состоит из двух полей: поля индексов (номера колонки или номера класса) и поля меток. Так как кэш-память организована в виде 128 классов, то разряды 20—26 определяют поле индексов, а разряды 11—19 — поле меток.

Кэш-память данных организована в виде 128 классов и двух групп ассоциативных кэшей: группы 0 и группы 1 (этот кэш можно представить в виде 128 колонок и двух строк). Логически каждая группа разделена на две части: массив индексов (массив меток) и массив данных. Каждая группа

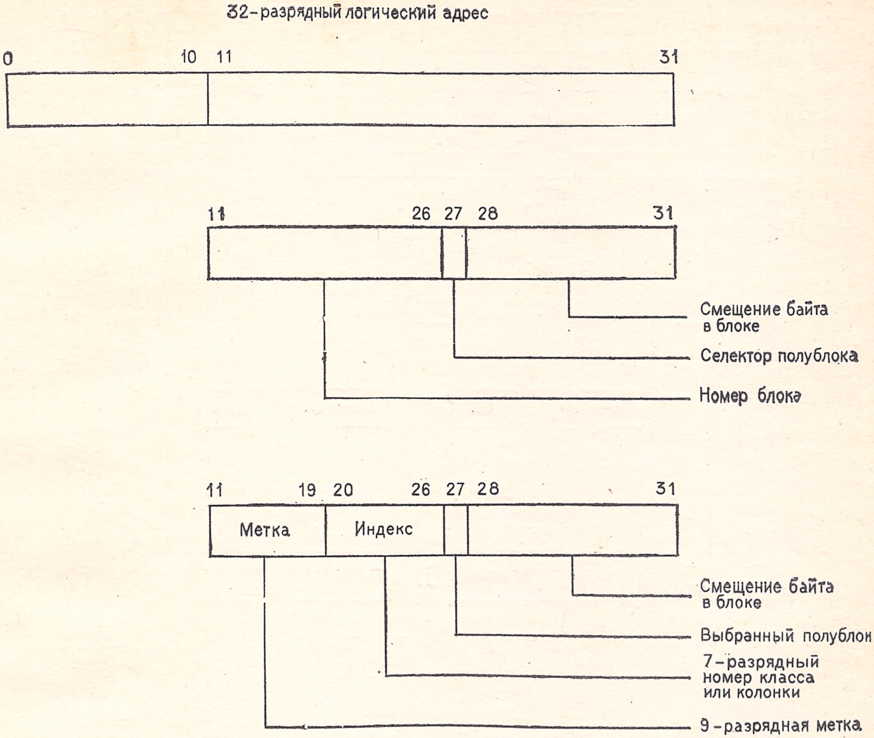


Рис. 7.12. Схема адресации в модели 155 системы IBM/370.

массива индексов содержит 128 записей (по одной на класс), и поэтому ей требуется семиразрядный адрес индекса. Каждая группа массива данных также содержит 128 записей. Однако поскольку каждое сообщение массива данных является блоком данных (двумя полублоками данных), то бит признака полублока (разряд 27) присоединяют к полю индексов (разрядам 20—26), чтобы сформировать адрес индекса для массива данных.

Каждая запись массива индексов состоит из трех полей: поля меток (разряды 11—19) физического адреса, двух разрядов опознавания (один для старшего полублока, а другой для младшего полублока) и разряда контроля ОК. Бит контроля ОК показывает, что соответствующие позиции в буферном и индексном массивах работают правильно (рис. 7.13). Основная память также делится на 128 классов, причем число блоков в каждом классе зависит от емкости основной памяти.

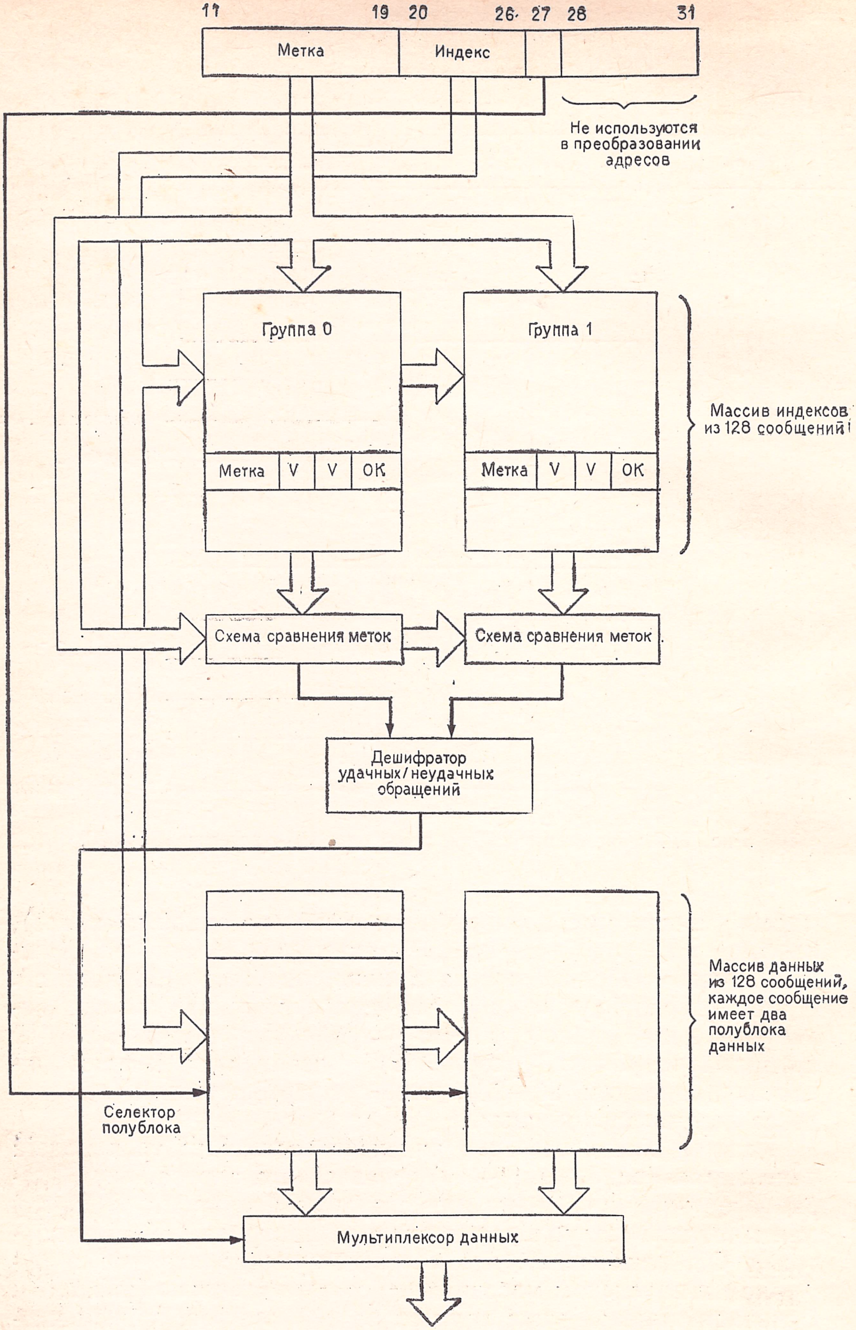


Рис. 7.13. Кэш-память в модели 155 системы IBM/370 (128 классов, два ассоциативных кэша).

Работа кэш-памяти

Кэш-память состоит из полублоков; для выборки используются индексы, метки и указатели полублоков. Устройство управления кэш-памятью содержит также регистр для определения того, к какой группе ячеек кэш-памяти следует обращаться.

Индексное поле исполнительного адреса (разряды 20—26) определяет ячейку в обеих группах индексного массива. Выбранные метки из обоих индексных массивов сравниваются с полем меток исполнительного адреса, сравниваются также соответствующие разряды признака полублока. Если есть соответствие между полем меток и признаком полублока исполнительного адреса с какой-либо строкой из индексных массивов, считается, что вызов данных принят, и запрашиваемый полублок вводится из кэш-памяти в процессор.

Если запрашиваемого признака и метки в кэш-памяти нет, то нужный полублок выбирается из основной памяти, засылается в процессор, а соответствующая ячейка кэш-памяти обновляется при помощи следующего алгоритма [К6].

1. Если в обеих группах имеется место для соответствующего блока, то, в какую группу поместить запрошенный блок данных, определяет регистр.

2. Если соответствующий блок в одной из групп занят, а в другой группе свободен, то блок помещается в свободную группу.

3. Если у соответствующих блоков в обеих группах занят один из полублоков, ни один из блоков не считается свободным, а какую группу выбрать, определяет регистр.

Кэш-память обновляется только в том случае, если индекс и метка изменяемого полублока соответствуют одному из двух индексных входных сообщений для этой колонки.

Буферное устройство управления кэш-памятью работает с помощью регистра, битов опознавания и битов правильности ОК. Когда запрашивается одна группа данного блока в данной колонке кэш-памяти, то регистр устанавливается для индикации другой группы (для индикации множества, запрошенного последним). Кроме того, при сбросе системы входные данные всех индексных массивов устанавливаются в нуль, сбрасываются все биты опознавания и устанавливаются все биты правильности ОК входных индексов данных. При отказе какого-либо элемента конкретного буфера бит правильности такого блока сбрасывается и формируется контрольное прерывание машины. У системы также имеется возможность отключения кэш-памяти либо выключателем на лицевой панели ЭВМ, либо микропрограммой управления,

что сопровождается некоторым ухудшением производительности.

В системе 370 модели 155 для кэш-памяти используется сквозная запись с целью уменьшения числа передач между основной памятью и кэш-памятью.

Кэш-память модели 165 системы 370

Быстродействующая кэш-память использовалась также в модели 165 системы IBM/370, которая представляет собой высокопроизводительную модель, созданную для универсальных научных и практических приложений, с временем машинного цикла процессора 80 нс и временем цикла основной памяти 2 мкс (с выборкой восьми байтов). Шина данных в процессоре и при обмене между процессором и памятью содержит восемь байтов. Однако чтобы достигнуть более высокой производительности, основная память используется с четырехкратным расслоением, а процессор спроектирован таким образом, что он может запрашивать память каждый машинный цикл. Различие в скорости работы центрального процессора и основной памяти (различаются в 25 раз) сглаживается быстродействующей кэш-памятью.

Кэш-память в модели 165 имеет емкость 8 или 16 Кбайт и время выборки 80 нс. Процессор может получить из этой памяти 8 байт данных за два машинных цикла, или за 160 нс, и 16 байт данных за 240 нс.

И основная память, и кэш-память данных разделены на блоки по 32 байт. Блок емкостью 32 байт является основной единицей обмена информацией между основной памятью и кэш-памятью.

Организация кэш-памяти

Система IBM 370 формирует 32-разрядный исполнительный адрес, но так как модель 165 позволяет работать с максимальным адресным пространством объемом 16 Мбайт, то она использует младшие 24 разряда (биты 8—31), как показано на рис. 7.14 Этот исполнительный адрес состоит из трех полей: разряды 29—31 определяют байт начального смещения внутри двойного слова, разряды 27 и 28 — конкретное двойное слово в блоке, а разряды 8—26 — адрес блока. Адрес блока в свою очередь состоит из двух полей: поля индексов и поля меток.

Длина поля индексов зависит от емкости кэш-памяти и ее организации. Для кэш-памяти емкостью 8 Кбайт, организованной в виде 64 колонок, разряды 21—26 (6 бит) служат полем индексов (номером колонки). Для буферной памяти

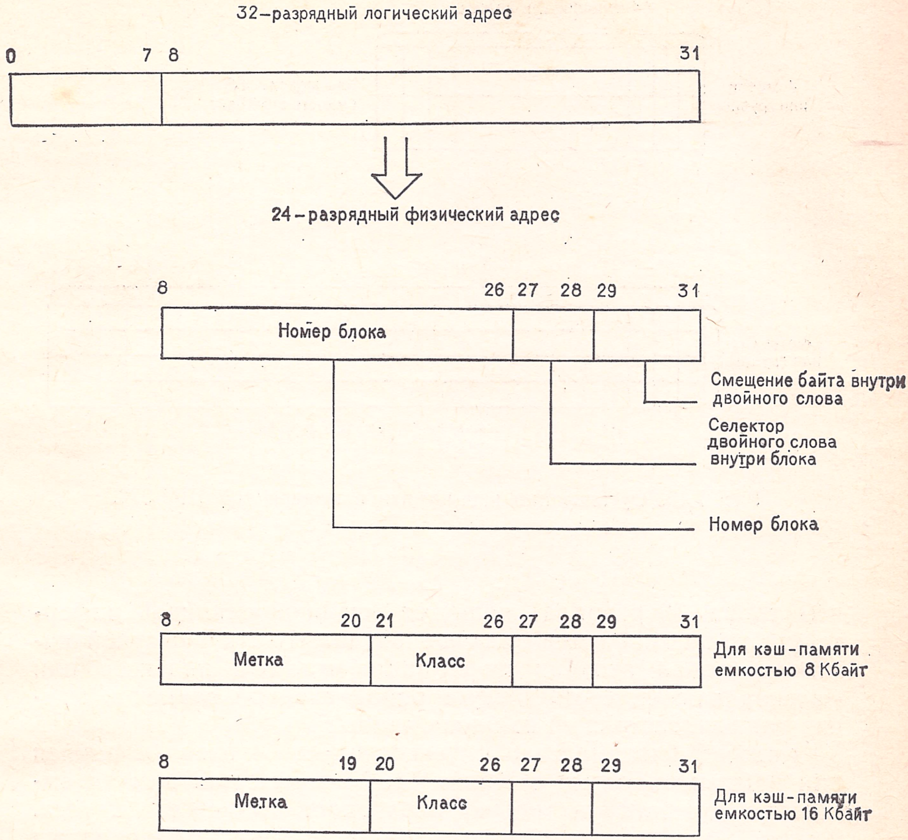


Рис. 7.14. Схема адресации в модели 165 системы IBM/370.

емкостью 16 Кбайт, организованной в виде 128 колонок, разряды 20—26 (7 бит) служат полем индексов. Для буферной памяти емкостью 8 Кбайт разряды 8—20 служат метками, а для памяти емкостью 16 Кбайт метками служат разряды 8—19.

Обе буферные памяти — емкостью 8 и 16 Кбайт — организованы в виде четырех групп ассоциативной кэш-памяти: группы 0, 1, 2 и 3. Информация этих четырех групп может быть представлена в виде 4 строк. Итак, кэш-память наглядно можно рассматривать либо как 64 колонки по 4 строки (для 8 Кбайт), либо 128 колонок по 4 строки (для 16 Кбайт) (рис. 7.15). Каждая группа (строка) логически состоит из трех частей: массива индексов (адресных меток), перемещаемого массива и массива данных. Так как блок (четыре двой-

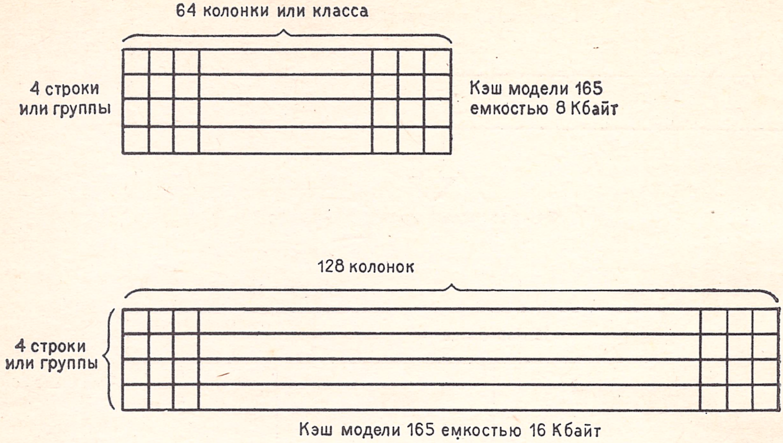


Рис. 7.15. Организация кэш-памяти в модели 165 IBM/370.

ных слова, 32 разряда) является основной единицей информации при обмене между основной памятью и буферной памятью, любая входная информация массива индексов или массива данных соответствует одной входной записи в массив данных, состоящей из блока данных.

Каждый блок массива индексов имеет 64 входные записи для памяти емкостью 8 Кбайт (128 записей для памяти емкостью 16 Кбайт) и требует 6 разрядов для своего адреса. Каждый блок массива данных также состоит из 64 входных записей (128 записей для памяти емкостью 16 Кбайт). Однако, поскольку каждая входная запись состоит из четырех двойных слов, каждая группа массива данных имеет объем 56 двойных слов и требует 8 разрядов для указания адреса. Разряды 27 и 28 физического адреса присоединены к разрядам 21—26 (разрядам 20—26 для буферной памяти емкостью 16 Кбайт) для формирования адреса индекса массива данных (рис. 7.16).

Каждая колонка в буфере данных соответствует перемещаемому массиву, состоящему из четырех записей, который управляет работой буферной памяти по перемещению. Когда процессор обращается к конкретному блоку буферной памяти, этот блок размещается в верхней части списка приоритетов соответствующей колонки, а когда конкретный блок требуется переслать в другую колонку, для перемещения выбирается входная информация, находящаяся внизу списка именно этой конкретной колонки.

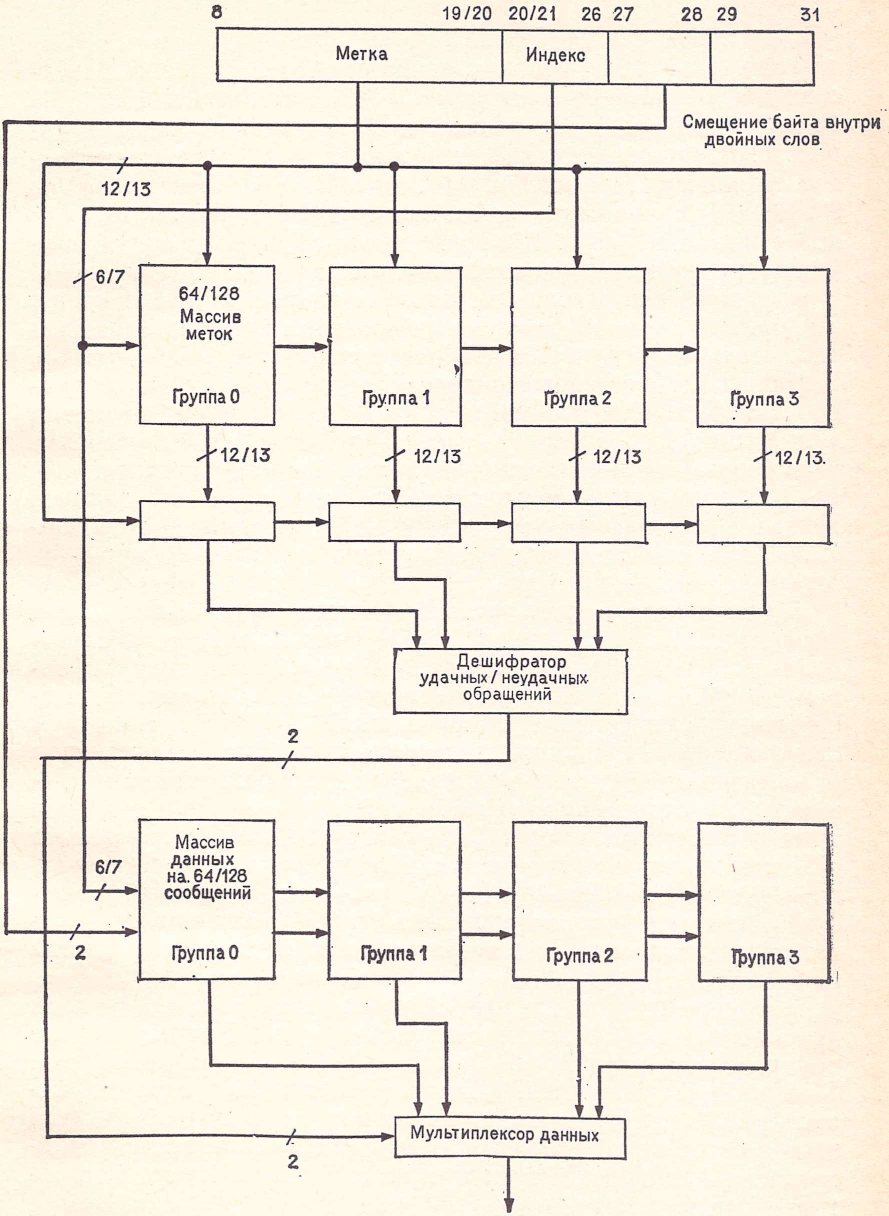


Рис. 7.16. Организация кэш-памяти данных модели 165 системы IBM/370 (кэш емкостью 8 или 16 Кбайт).

Работа кэш-памяти

Индексное поле исполнительного адреса массива индексов, разряды 21—26 (разряды 20—26 для памяти емкостью 16 Кбайт), определяет ячейку во всех группах массива индексов. Выбранные метки всех четырех групп массива индексов сравниваются с меткой исполнительного адреса, разряды 8—20 для кэш-памяти емкостью 8 Кбайт (разряды 8—19 для кэша емкостью 16 Кбайт). Если имеет место соответствие между полем меток исполнительного адреса и меткой любой из групп, то считается, что ответ на запрос данных получен и запрошенное двойное слово из соответствующего блока соответствующей группы кэш-памяти передается в процессор. Истинное двойное слово, определяемое байтом смещения, находящимся внутри двойного слова, может быть присоединено к выходной информации массива индексов, чтобы выбрать любой отдельный байт из требуемых данных. Отметим, что, так как один кэш используется и для команд, и для данных, данные в кэш-памяти могут быть либо данными, затребованными процессором, либо даже командой, которая может быть загружена в буфер команд.

Однако, если индекс запрашиваемого блока не найден в массиве индексов, соответствующий блок данных следует выбрать из основной памяти, передать в процессор, а также запомнить в кэш-памяти для будущего использования. В кэш-памяти, примененной в машине IBM 360/85, используется схема сквозной загрузки приходящих данных в процессор (т. е. конкретное двойное слово, запрошенное процессором, — это первое слово, которое выбирается из основной памяти и засылается в процессор). Остальные двойные слова блока выбираются из основной памяти и засылаются в процессор последующими циклами. Блоку, размещаемому внизу массива перемещаемого конкретного столбца, приводится в соответствие этот новый блок данных, и обновляется массив индексов.

В кэш-памяти также используется схема сквозной записи для обновления данных в основной памяти. Данные в кэше обновляются только в части записываемых разрядов. При отсутствии записи обновляется только основная память. Так как при сквозной записи в основной памяти всегда содержится достоверная копия информации, нет необходимости любую информацию передавать из кэша в основную память.

Кроме моделей 155 и 165 кэш-память была введена также в модели 168, 168-3 и 195. Фактически концепция кэш-памяти использовалась почти во всех крупных моделях. В табл. 7.1 систематизированы размеры и быстродействие кэш-памятей

для всех моделей систем 360 и 370 [B13]. Идея кэш-памяти применена также в моделях серии 303X.

Таблица 7.1. Время цикла работы процессора и данные кэш-памяти для систем 360 и 370

Модель	Время цикла процессора, нс	Время цикла основной памяти, нс	Размер кэша	Размер блока	Время цикла кэш-памяти	Тип записи	Ассоц. выборки
360/85	80	960	16—32	16×4	80—100	ST[а]	16
360/195	54	756	32	8×8	54—112	ST	4
370/155	115	2070	8	16	115—230	ST	2
370/155 II	115	2070	8	16	110—230	ST	2
370/158 и	115	1035R	8	16	115—230	ST	2
158-3	115	920W					
370/165 и	80	2000	8—16	8×4	80—110	ST	4
165 II	80	2000	8—16	8×4	80—160	ST	4
370/168 и	80	320	8—16	8×4	80—160	ST	4—8
168-3	80	320	32	8×4	80—110	ST	8
370/195	54	756	32	8×8	54—162	ST	4
3031	115	345	32	8×4	115—230	ST	8
3032	80	320	32	8×4	115—230	ST	8
3033	57	285	64	8×8	57—114	ST	16
3033N	57	285	16	8×8	57—114	ST	8
3033S	57	285	0,5	8×4	57—114	ST	8
3081	26	312	32	8×16	26—52	SW[б]	4
4331-1	300—1600	900R/1300W	—				
4331-2	200—1600	2600R/3100W	8	4×16	200	SW	4
4341-1	150—300	2400	8	8×8	225	SW	4
4341-2	120—240	1440	16	16×4	120R 180W	SW	8

[а] ST — сквозная запись.
[б] SW — простой свопинг.

Кэш-память ЭВМ IBM 3033

Быстродействующая кэш-память относительно большой емкости использована в модели 3033. ЭВМ IBM 3033 является одной из универсальных машин семейства 370; она может работать с основной памятью емкостью 32 Мбайт, имеет цикл работы процессора 57 нс и выполняет около 5 млн. операций в секунду. Конструкция ЭВМ 3033 напоминает модель 168 системы 370.

ЭВМ 3033 имеет быстродействующую кэш-память емкостью 64 Кбайт и для команд, и для данных. Эта большая быстродействующая память является одной из главных причин большого увеличения производительности ЭВМ.

Кэш имеет тот же цикл работы, что и процессор, равный 57 нс.

Цикл работы основной памяти в ЭВМ 3033 равен 285 нс. Основная память разделена на восемь логических элементов памяти (ЛЭП) и расходится по восьми шинам. Двойное слово (восемь байт) является основной единицей данных, поступающих из каждого ЛЭП; восемь двойных слов (блок из 64 байт) могут быть вызваны из восьми ЛЭП одновременно.

Различие в скорости работы процессора и основной памяти (в 5 раз) устраняется с помощью быстродействующей кэш-системы. Емкость кэша для ЭВМ 3033 равна 64 Кбайт, быстродействующий кэш используется и для данных, и для команд; время цикла 57 нс. Кэш спроектирован таким образом, что выборка может быть осуществлена каждый машинный цикл. Процессор может получить двойное слово из кэш-памяти за два цикла (114 нс), а 16 байт — за 171 нс.

И основная память, и кэш данных разделены на блоки по 64 байт. Блок емкостью 64 байт является основной единицей обмена данными между основной памятью и кэш-памятью.

Организация кэш-памяти

Так как модель 3033 может работать с максимальным адресным пространством емкостью 32 Мбайт, в ней используются младшие 25 разрядов исполнительного адреса (разряды 7—31), как показано на рис. 7.17. Этот исполнительный адрес состоит из трех полей: разряды 29—31 определяют смещение внутри двойного слова, разряды 26—28 — конкретное двойное слово внутри блока, а разряды 7—25 (19 бит) — адрес блока. Адрес блока в свою очередь состоит из двух полей.

Так как кэш-память с ассоциативной выборкой для модели 3033 организована в виде 64 колонок, в каждой из которых размещено 16 строк, то разряды 20—25 (6 бит) исполнительного адреса служат индексным полем, а разряды 7—19 (13 бит) — полем меток. Каждая строка кэш-памяти состоит из трех частей: массива индексов (адресных меток), массива перемещения и массива данных. Так как блок (восемь двойных слов, 64 байт) является основной единицей информации, циркулирующей между основной памятью и кэш-памятью, каждая единица входной информации массива индексов кэш-памяти соответствует единице входной информации массива данных, состоящей из блока данных.

Каждая группа массива индексов содержит 64 сообщения и требует шести разрядов для указания адреса каждого со-

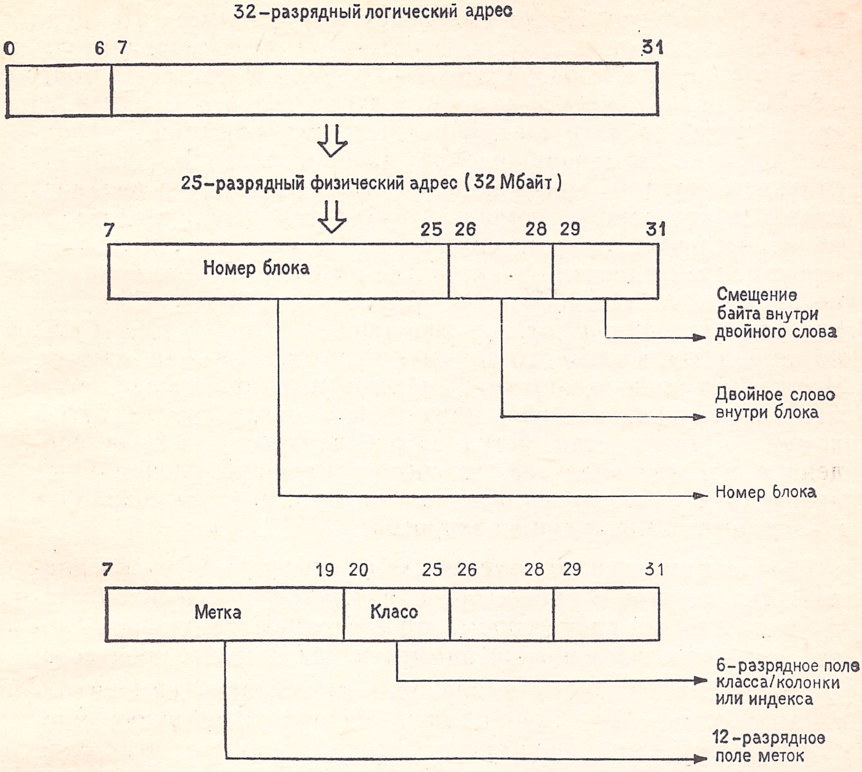


Рис. 7.17. Схема адресации в IBM 3033.

общения. Каждая группа массива данных также содержит 64 сообщения. Однако, поскольку каждое сообщение из массива данных состоит из восьми двойных слов, каждая группа массива данных имеет объем 512 двойных слов и для ее адресации требуется девять разрядов. Разряды 20—28 служат адресом индекса для массива данных.

Каждой колонке в кэше данных сопоставлен массив перемещения, содержащий 16 записей. Этот массив управляет работой кэша по перемещению данных. Когда конкретный блок кэша вызывается процессором, он ставится наверху списка для конкретной колонки, а блок, находящийся внизу списка для этой колонки, выбирается для замещения.

Работа кэш-памяти

Поле индексов исполнительного адреса для массива индексов (разряды 20—25) выделяет ячейку во всех группах

массива индексов. После выделения меток из всех 16 групп массива индексов они сравниваются с меткой исполнительного адреса, т. е. с разрядами 7—19. Если имеет место соответствие между меткой исполнительного адреса и одной из меток какой-либо группы, считается, что необходимые данные найдены, и запрашиваемое двойное слово из соответствующей группы пересылается в процессор. Адрес двойного слова, выделенный с помощью байта смещения, взятого от исполнительного адреса, может быть объединен с выходной информацией массива индексов для выборки любого конкретного байта из затребованных данных (рис. 7.18). Отметим, что так как один и тот же кэш данных используется и для команд, и для данных, то данные, полученные из кэш-памяти, могут быть либо данными, необходимыми процессору, либо даже командой, которая может быть загружена в буфер команд. Однако если метка затребованного блока не найдена в массиве индексов, то соответствующий блок данных необходимо получить из основной памяти, а затем сохранить в кэш-памяти для будущих запросов.

В кэш-памяти используется схема сквозной загрузки приходящих данных в процессор, т. е. конкретное двойное слово, запрошенное процессором, это и есть первое двойное слово, выбранное из основной памяти и посланное в процессор. Остальная часть блока выбирается из основной памяти последовательно. Блок кэш-памяти, находящийся внизу массива перемещения конкретной колонки, принимает этот блок данных, а массив индексов соответствующим образом обновляется.

В кэш-системе применяется также стратегия сквозной записи для обновления данных кэш-памяти и основной памяти. Кэш данных обновляется только при операциях записи. При отсутствии записи обновляется только основная память. Так как при схеме сквозной записи всегда имеется достоверная копия информации в основной памяти, нет необходимости передачи какой-либо информации из кэша в основную память.

Основную память процессора для ЭВМ 3033 можно также рассматривать как логически разделенную на 64 класса с числом блоков в классе, зависящим от емкости памяти. Для основной памяти емкостью 4 Мбайт число блоков в классе будет равно 1024, для памяти в 32 Мбайт в одном классе будет находиться 8K блоков. Так как кэш-память организована в виде 16 ассоциативных групп, то любой блок любого конкретного класса основной памяти может быть сохранен в любой из 16 групп соответствующего класса.

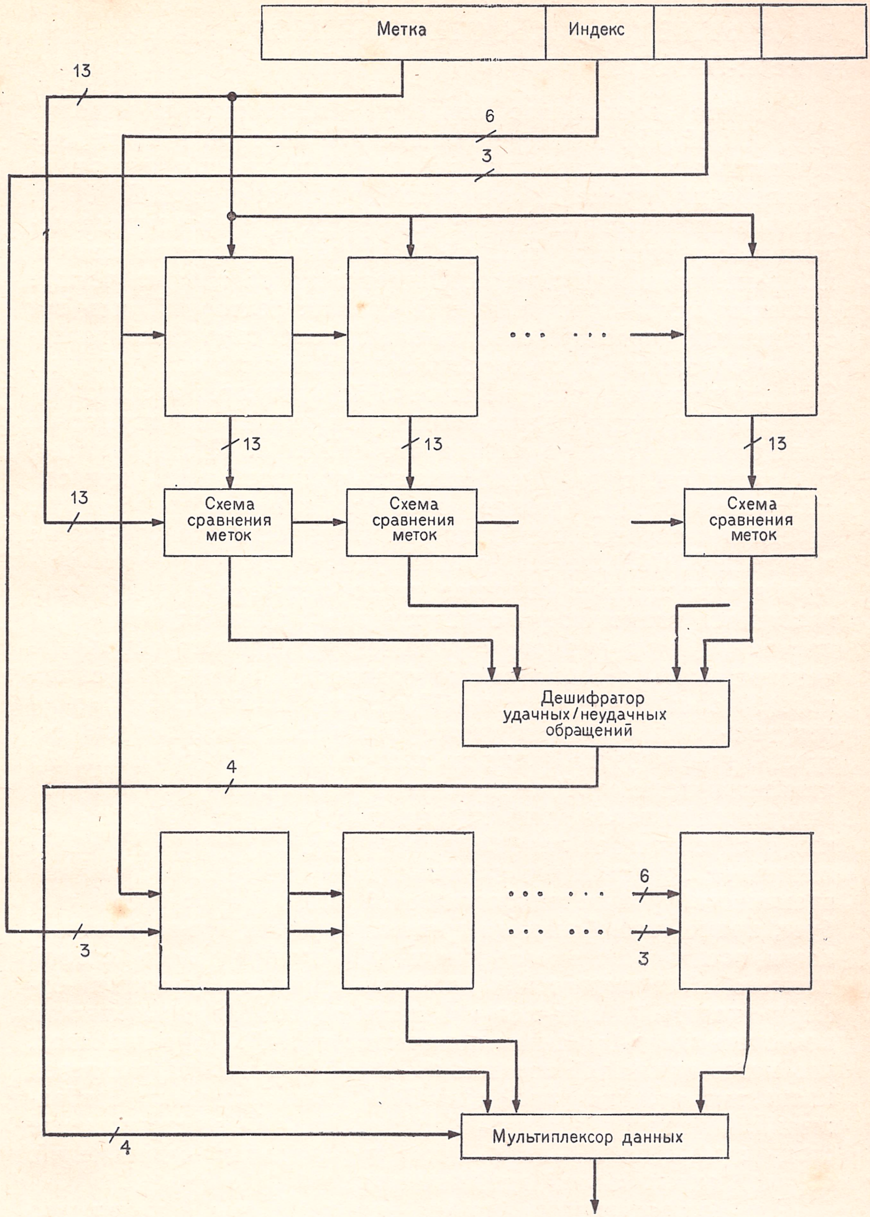


Рис. 7.18. Организация кэш-памяти данных в IBM 3033.

7.3. ДРУГИЕ ВОПРОСЫ, ОТНОСЯЩИЕСЯ К КЭШ-ПАМЯТИ

Удачное воплощение концепии кэш-памяти в моделях IBM 360/85, 195 и в серии машин 370 вызвало в конце шестидесятых и семидесятых годов значительный интерес к дальнейшему применению и исследованию этих идей. В 1969 г. Конти в превосходном обзоре [C1] рассмотрел различные методы распределения памяти и различные алгоритмы записи и перемещения информации. Он детально рассмотрел четыре основных метода распределения памяти: секторный, или постраничный, прямого отображения, полностью ассоциативного отображения и группового ассоциативного отображения. Эти четыре метода все еще широко используются при отображении информации в кэш-памяти. Он указал преимущества и недостатки каждого метода отображения. Конти показал, что схема секторного отображения при разбиении каждого сектора на блоки, близкие по объему к оптимальной длине передачи и снабженные битами достоверности (используемые в модели 360/85), имеет то преимущество, что при этом требуется относительно меньшее число меток, и в результате появляется возможность одновременного сравнения со всеми метками. Метод прямого отображения позволяет исключить необходимость ассоциативного сравнения и ведет к самой простой аппаратуре и к самой быстрой из возможных выборок кэш-памяти за счет увеличения числа конфликтных ситуаций при выборке блока из кэш-памяти. Схема полностью ассоциативного отображения, хотя и является лучшей для минимизации числа конфликтных ситуаций при выборке блоков и обеспечивает наибольшую вероятность удачного обращения, не представляет практического интереса на фоне современного уровня технологической базы. Такая схема требует, кроме того, бо́льшего времени выборки (из-за длительности последовательного поиска) по сравнению со схемой прямого отображения или даже секторной организацией буфера (который позволяет избежать полной ассоциативности при очень незначительном количестве меток). Конти показал, что метод групповой ассоциации имеет преимущества и перед полностью ассоциативной схемой, и схемой прямого отображения и, весьма возможно, обеспечит наилучшее соотношение стоимости и производительности в пределах ограничений, обусловленных современной технологией.

Что касается алгоритмов записи, то Конти показал, что алгоритм сквозной записи удобен для схем секторного отображения информации. Кэш с секторным отображением информации не очень удобен для свопинга, так как не исключено, что перемещаемый сектор будет содержать много бло-

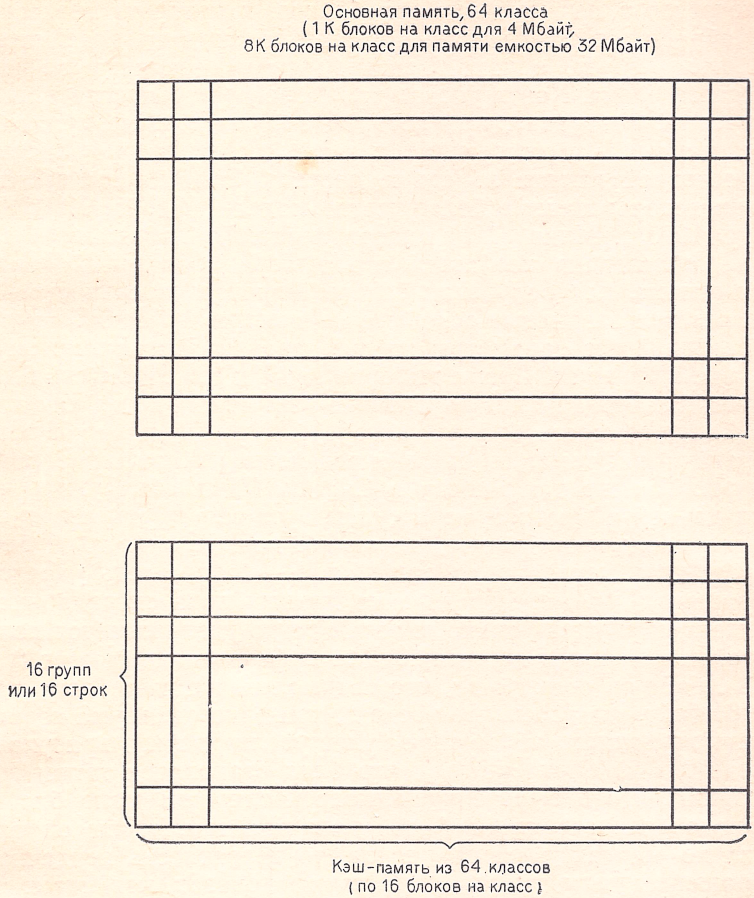


Рис. 7.19. Основная память и организация кэш-памяти в IBM 3033.

ков, требующих перемещения во внешнюю память, что в таком случае приведет к очень большому времени перекачки и на долгое время займет память следующего уровня иерархии. В ЭВМ 360/85 в кэш-памяти с секторным отображением использовалась схема сквозной записи. Схема свопинга отвечает требованиям по быстродействию в других трех методах отображения информации. Они обеспечивают более высокую производительность, чем схема сквозной записи.

Конти пояснил также работу различных алгоритмов перемещения, начиная с метода полностью случайного перемещения и кончая очень сложными методами. Выяснилось, что алгоритм, дающий удовлетворительные результаты, за-

ключается в перемещении тех блоков, к которым процессор не обращался самое продолжительное время. Такая схема работы требует, однако, вводить разряд «активности» для каждого блока. Конти указал также на преимущество метода «сквозной загрузки», при котором запрашиваемые процессором данные, отсутствующие в кэше, выдаются в процессор из основной памяти в первую очередь. Эти методы позднее были использованы в большинстве моделей фирмы IBM.

В 1970 г. Миид [M1] описал иерархию памяти в общем виде, подтвердив необходимость выбора правильной иерархии памяти для получения большей производительности. Он рассмотрел различные основные параметры буферной памяти и разделил все параметры на две основные категории: внутренние и внешние. Время выборки кэша и его стоимость были классифицированы как внутренние параметры, тогда как длина блока, емкость буфера, алгоритм управления и скорость передачи информации были определены как внешние параметры. Так же как Гибсон и Сиссон, Миид обнаружил, что для данной емкости буфера существует соответствующая оптимальная длина блока. Для любой данной емкости буфера вероятность неудачного обращения (1 — вероятность удачного обращения) по мере увеличения длины блока сначала имеет тенденцию к уменьшению, а затем резко увеличивается после того, как оптимум достигнут. Для буферов емкостью от 2 до 128 Кбайт он обнаружил, что оптимальная длина блока равна 64 байт. Казалось, что после достижения оптимума вероятность неудачного обращения будет увеличиваться при увеличении длины блока, потому что блок становился таким громоздким, что в буфере размещалось слишком мало блоков. Однако если емкость буфера увеличить так, чтобы он вместил то же самое число блоков, то вероятность неудачного обращения уменьшится. Миид также показал, что поток информации между буфером и следующим уровнем иерархии памяти определяется произведением вероятности неудачного обращения на длину блока, и, так как блок являлся основной единицей передачи данных, большая длина блока приводила к увеличению потока данных. Бóльшие блоки требовали бóльшей емкости буфера, чтобы сохранить в них то же число блоков, бóльшее время передачи информации и бóльшую производительность памяти ближайшего уровня иерархии.

Миид показал, что для данных емкости буфера, длины блока и алгоритма существует соответствующее распределение запросов для набора программ. Бóльшая емкость буфера, по-видимому, обеспечивает бóльшую вероятность удачного обращения. Он отметил, что емкость буфера должна выбираться на основе анализа производительности и соотношения

«стоимость — производительность», и показал, что емкость буфера для больших систем должна достигать по крайней мере 8 Кбайт, чтобы получалась достаточно высокая вероятность удачного обращения (близкая к 99 %). Кроме того, так как выяснилось, что емкость буфера не зависит от емкости памяти, введение иерархии памяти с кэшами казалось более предпочтительным, чем одноуровневая память большой емкости. Миид энергично выступал за необходимость проектирования таких систем иерархии памяти, которые позволяют достигнуть наилучшего соотношения производительности и стоимости всей системы памяти в целом, а также сформулировал правила создания двух- и трехуровневых систем иерархии памяти.

В 1971 г. Миид [М2] подробно рассмотрел вопросы проектирования, реализации логики и алгоритмы управления для полностью ассоциативной кэш-памяти и кэш-памяти с групповой ассоциацией при емкостях кэшей, колеблющихся от 4 до 16 Кбайт. Он исследовал машину с адресным полем в 16 Мбайт. И кэш-память, и основная память были разделены на блоки длиной 64 байт; каждый блок состоял из восьми слов по 8 байт. Миид предложил несколько путей выполнения ассоциативных схем, таких, как список магазинного типа или простая схема со счетчиком. В конце концов он остановился на схеме со счетчиком числа блоков. Кэш состоял из массива меток, массива данных с произвольным доступом, таблицы подсчета и логики, управляющей перемещением. Миид настолько детально рассмотрел вопросы проектирования, что оказалось возможным сравнить два подхода. Он обнаружил, что простой двухгрупповой ассоциативный контроллер имеет задержку, равную приблизительно трем четвертям задержки полностью ассоциативной версии (на 25 % быстрее). Миид показал также, что, в то время как для кэш-памяти меньшей емкости (4 Кбайт, 64 блока) стоимость блока управления для обоих вариантов приблизительно одинакова, для кэш-памятей бóльшей емкости полностью ассоциативная схема кэш-памяти приблизительно в четыре раза дороже, чем схема с групповой ассоциацией. Миид предсказал более широкое использование полностью ассоциативных алгоритмов по мере развития технологии и появления памяти, адресуемой по содержимому, все большей емкости.

В 1972 г. Миид [М3] снова сделал попытку объяснить смысл кэш-памяти и привел основные правила проектирования хорошей кэш-памяти. Он исследовал возможность использования кэшей для мультипроцессорных и мультипрограммных систем. Миид указал на необходимость выбора необходимой емкости кэш-памяти и поддерживающей ее памяти с помощью соответствующего анализа производитель-

ности и стоимости. Поддерживающая память характеризовалась как имеющая фиксированную емкость при переменных стоимости и производительности, тогда как кэш-память представлялась как память переменной емкости при фиксированных стоимости и производительности. Сравнивая соотношения стоимости и производительности для нескольких гипотетических машин, Миид показал, что кэш-память в больших вычислительных машинах является более эффективной по стоимости, чем в малых системах. В больших системах стоимость кэша незначительна по сравнению с общей стоимостью машины, тогда как в малых машинах стоимость кэш-памяти составляет значительную часть общей стоимости.

Для систем средних размеров суждение о включении кэша может быть сделано, только когда кэш-память и память следующего уровня иерархии вместе будут стоить меньше, чем одна основная память с быстродействием, достаточным для того, чтобы обеспечить такую же производительность. Миид распространил концепцию кэш-памяти на микропрограммные, многопроцессорные и мультипрограммные системы, а также системы виртуальной памяти. Он был убежден, что достижения технологии позволят создать быстродействующие биполярные кэши (с временем выборки 16—20 нс), работающие с основной памятью на МОП-структурах, имеющей быстродействие 240—320 нс, а это сделает концепцию кэш-памяти применимой во многих системах.

Удачное воплощение концепции кэш-памяти в моделях 85 и 195 системы IBM/360 и в различных моделях системы IBM/370, таких, как 155 и 165, вызвало значительный интерес к системам иерархии памяти. В начале семидесятых годов появился интерес к внедрению многоуровневых систем иерархии памяти с упором на автоматическое оптимальное управление иерархией памяти и выбор соответствующей технологии для оптимального построения систем памяти для ЭВМ.

Чтобы обеспечить создание оптимальной многоуровневой системы иерархии памяти, Мэтсон и др. [М8] разработали в 1970 г. метод, известный как стековая обработка для измерения производительности широкого класса многоуровневых систем памяти с выборкой страниц по запросу, использующих множество схем отображения и однопроходных алгоритмов перемещения информации в адресном пространстве. Они нашли, что общепринятые методы моделирования являются слишком медленно действующими, чтобы можно было использовать их для оптимального выбора технологии, внедрения этой технологии и организации потока данных между различными уровнями иерархии. Метод стековой обработки они определили как метод эффективной оценки соотношения

«стоимость — производительность» для широкого класса систем иерархии памяти и добились того, что вычисления при моделировании вероятности удачных обращений выполнялись в тысячу раз быстрее, чем прежде. Для многоуровневой системы иерархии памяти необходимо принимать во внимание следующие параметры проектирования: 1) стоимость и скорость работы процессора, 2) аппаратная база построения буфера; 3) аппаратная база построения памяти; 4) емкость буферной памяти; 5) размер страницы; 6) число блоков на странице; 7) число классов; 8) размер буферного страничного блока в классе; 9) алгоритм перемещения данных; 10) метод управления памятью.

Мэтсон показал, что при заданной емкости буферной памяти и данном числе классов вероятность неудачного обращения к памяти уменьшается с увеличением размера страницы и существует минимальная вероятность неудачного обращения при заданной емкости буфера. После того как этот минимум достигнут, вероятность неудачного обращения увеличивается с ростом размера страницы. Кроме того, оказывается, что для заданного размера страницы вероятность удачного обращения увеличивается с ростом емкости буфера. По данным Мэтсона, емкость в 64 или 128 байт является оптимальной для блока.

Мэтсон использовал данные по вероятностям удачного обращения совместно с технологическими параметрами, временными и стоимостными характеристиками для того, чтобы вычислить «стоимостную производительность», измеряемую в долларах на секунду за одно обращение для различных систем памяти. Технологический параметр отражал стоимость процессора и технологию, используемую при построении буфера и памяти. Принятые допущения по стоимости и временным характеристикам включали время поиска оглавления, время обновления списка приоритетов, время выборки данных, цикл работы памяти, время перемещения страницы, время контроля и корректировки списка по четности, время обновления оглавления и среднее время ответа системы памяти. Суммарная стоимость системы включала стоимости процессора, буфера, справочного файла, списка приоритетов и памяти.

Лин и Мэтсон [L7] использовали метод стековой обработки, чтобы сравнить «стоимостные производительности» двух- и трехуровневой иерархии и иерархии, использующей методы произвольной выборки и устройств последовательного доступа. Они обнаружили, что «стоимостная производительность» системы очень сильно зависит от числа линий в магистрали, связывающей различные уровни иерархии. Отношение времени выборки основной памяти к времени выборки памяти

следующего уровня иерархии в меньшей степени влияет на «стоимостную производительность» системы. Они нашли также, что для заданной системы иерархии существует оптимальное отношение между временами выборки, которое увеличивается с ростом емкости памяти следующего уровня иерархии и заметно уменьшается с уменьшением числа линий в магистрали. Было установлено, что до тех пор, пока отношение времени доступа к буферу к времени доступа к памяти следующего уровня иерархии не превышает величины порядка 50, двухуровневая система иерархии памяти имеет лучшую «стоимостную производительность», чем трехуровневые системы иерархии. Лин и Мэтсон также исследовали возможность использования памяти на магнитных доменах в качестве памяти следующего уровня иерархии и пришли к заключению, что для емкости не выше 1 Мбайт память с произвольным доступом более предпочтительна по сравнению с устройствами последовательного доступа, а для емкостей 4 и 16 Мбайт предпочтительнее системы с последовательным доступом, снабженные регистрами сдвига длиной 256 и 1024 разряда соответственно.

В 1972 г. Хэлтфилд с сотр. [H4] провели исследование по определению оптимальной длины страницы для системы виртуальной памяти и пришли к заключению, что общие допущения, принятые для систем виртуальной памяти (т. е. верхнюю границу времени выборки, время доступа, размер страницы следует уменьшать), не всегда подходят. С помощью различных экспериментов было установлено, что бóльшие размеры страницы обеспечивают лучшую производительность при работе с программами, которые используют массивы, компактно расположенные в пространстве памяти. Был сделан вывод, что наблюдаемую зависимость между содержанием программы на странице и алгоритмом перемещения страницы пока нельзя убедительно объяснить.

В 1972 г. Чу и Опдербек [C5] разработали новую стратегию перемещения, названную ими алгоритмом перемещения страницы с помощью прерывания, вызываемого отсутствием страницы в памяти. В этом алгоритме в качестве основного параметра использовалась частота прерывания из-за отсутствия страницы в памяти. Такая схема не требовала предварительной информации о работе программы. Она распределяла память в соответствии с динамически меняющимися запросами памяти на каждом отрезке программы. Получив разнообразные результаты моделирования, Чу и Опдербек нашли, что эффективность этого алгоритма выше, чем алгоритма LRU, и сравнима с алгоритмом перемещения рабочих программ Деннинга [D1]. Кроме того, введение алгоритма перемещения PFF, по-видимому, проще, чем внедрение алго-

ритма LRU, и намного проще, чем внедрение в систему алгоритма перемещения рабочих программ.

В 1972 г. Каплан и Уиндер [K2] также рассмотрели проблему проектирования вычислительной системы, использующей кэш-память с иерархией, и предложили новый параметр эффективности работы кэш-памяти: $u = (1 - HR)r$. Этот параметр больше подходит к командам и определяет случаи неудачного обращения к памяти. Авторы очень верили в преимущества, даваемые использованием u в качестве кэш-параметра, так как u влияет на производительность и, что наиболее важно, не зависит от проектных параметров кэша (таких, как число линий в магистрали между процессором и кэш-памятью, метод выполнения команд и алгоритм выбора команды).

Они определили два новых параметра: u — долю неудачных обращений в расчете на одну команду, и k — стоимость избыточных обращений, возрастающую из-за ветвления программы. Использование параметра k позволило получить существенную независимость от взаимного влияния производительности и параметров проектирования кэш-памяти. Каплан и Уиндер представили новые данные по производительности, полученные при решении системой обработки данных численных примеров, таких, какие могли быть использованы разработчиком новой системы.

В 1973 г. Пом и др. [P4] исследовали возможность создания многоклассовой системы буферной памяти с изменяемым сегментным указателем и списком обновляемых данных, реализованным на быстродействующих ячейках памяти. Такая система была спроектирована в модульной форме так, что число слов в блоке, число блоков в классе и число классов могли быть легко изменены, что позволяло приспосабливать эту систему к нуждам пользователя. Они продемонстрировали, что использование буфера увеличило стоимость 500 000 байт памяти приблизительно на 8 %, уменьшив эффективное время цикла выборки в три раза.

В 1974 г. Агравал [A6] исследовал применимость концепции кэш-памяти к нетрадиционным вычислительным структурам, подобным системе SYMBOL-2R. Его целью было проанализировать влияние организации архитектуры нетрадиционной вычислительной системы на процесс проектирования кэш-памяти этой системы, чтобы понять, должна ли кэш-память в неоднородной системе разделения времени широкого назначения быть выполнена в виде однородной или неоднородной структуры, и определить, должна ли кэш-память быть разделена на одинаковые по объему части для различных классов или же объем кэш-памяти, отнесенный к классу, должен зависеть от нужд и требований этого класса.

Было показано, что для вычислительных структур, подобных системе SYMBOL-2R, на способ формирования адреса выборки и общее время работы с памятью различных специализированных процессоров сильно влияют организация памяти и метод управления системой виртуальной памяти. Экспериментальные данные показали, что каждому процессору требуется кэш-память со своими характеристиками, которые больше зависят от специализации данного процессора, чем от выполняемой системой рабочей программы. Отсюда он сделал вывод, что в целом вероятность удачного обращения к памяти не будет значительно зависеть от режима работы системы, задаваемого пользователем. Было обнаружено, что очень эффективной является концепция применения небольших специализированных кэш-памятей для многофункционального процессора систем, подобных SYMBOL-2R, ввиду того, что на время, когда происходил поиск информации с помощью указателя, память отключалась от процессора и, следовательно, не мешала работе других процессоров.

В 1974 г. Чоу [C8] рассмотрел создание оптимальной иерархии памяти как задачу геометрического программирования и построил весьма простую модель ее. Он принял, что функция вероятности удачного обращения может быть представлена степенными функциями, и вывел формулы, связывающие минимальное время выборки памяти с ее иерархией. Было показано, что оптимальное число уровней иерархии в памяти пропорционально логарифму объема системы памяти с коэффициентом пропорциональности, зависящим от технологических характеристик системы и величины вероятности удачного обращения к системе памяти. Чоу также показал, что оптимальные отношения стоимостей и емкостей устройств памяти, находящихся на соседних уровнях иерархии, и отношение времен выборки информации из запоминающих устройств представляют собой постоянные величины.

Для быстрой оценки технологических и структурных преимуществ той или иной системы иерархии памяти Гекси и Льюкс [G7] разработали модель, основанную на модели циклической организации вычислительных систем и их программного обеспечения. Их целью при создании этой модели было получить эффективный в вычислительном отношении алгоритм для оценки влияния на производительность изменений загрузки системы вычислительными задачами и узлов, составляющих иерархическую память.

Гекси в 1974 г. [G8] предложил использовать идею вычислений с помощью стека, предложенную Мэтсоном и др., для определения вероятностей удачных обращений для класса систем многоуровневых иерархий памяти, определяемых как

ступенчатые иерархии. Концепция ступенчатых иерархий, введенная Слутцем и Трейгером [S9], применима для произвольного числа уровней памяти, использующих различные длины блоков информации на разных уровнях иерархии, а также при многократном копировании содержимого одного и того же блока в системе.

В 1975 г. Пом и др. [P3] выделили четыре метода осуществления записи в память: 1) сквозная запись [C3], когда информация в основной памяти всегда обновляется при операциях записи; 2) простой свопинг (ПС), при котором информация переписывается снова в основную память при выводе ее из кэш-памяти; 3) флаговый свопинг (ФС), когда в основную память переписывается только измененная информация; 4) флаговый регистровый свопинг (ФРС), характеризующийся тем, что блок информации, который был изменен и который следует записать обратно, сначала переписывается в регистр, а затем записывается в память. Они получили соотношение между фактическим временем выборки информации при вычислениях и длительностью цикла работы памяти для различных случаев. Было показано, что алгоритм ФРС имеет преимущество перед всеми остальными алгоритмами. Ими было показано, что для буфера с быстродействием 50 нс и памятью следующего уровня иерархии с быстродействием 750 нс, а также для буфера с быстродействием 200 нс и памятью следующего уровня иерархии с быстродействием 1500 нс даже при вероятности удачного обращения, не превышающей 0,80—0,82, можно достигнуть четырехкратного увеличения быстродействия по отношению к скорости работы памяти более низкого уровня иерархии. Было также принято во внимание увеличение производительности за счет чередования адресов. Они рассмотрели две альтернативные структуры с четырьмя модулями памяти. В одной структуре к каждому модулю памяти низшего уровня иерархии с временем выборки 700 нс и временем цикла 1500 нс подключалась буферная память емкостью 500 слов и с быстродействием 200 нс. В другой структуре для передачи четырех слов блока памяти одновременно к четырем работающим модулям памяти подключался буфер емкостью 2000 слов. Эквивалентная фактическая длительность цикла для четырех независимо работающих модулей составляла 100 нс, а для системы с общим буфером она была равна 215 нс.

Было показано, что раздельная буферизация модулей основной памяти позволяет получить очень высокую общую производительность системы, недостижимую при использовании общего буфера. Стоимость, однако, такой системы выше, чем при применении общего буфера.

7.4. Кэш-память в мини-ЭВМ

В 1971 г. Белл и Кэсесент [B3] исследовали возможность использования небольших кэш-памятей для ЭВМ PDP-8/E для получения пятикратного повышения производительности. Эта ЭВМ имела емкость основной памяти 4К слов с циклом работы 1 мкс. Они продемонстрировали возможность получения фактической длительности цикла в 200 нс при использовании кэш-памяти с быстродействием в 100 нс. Было отмечено, что благодаря простоте реализации для мини-ЭВМ наиболее удобна простая схема прямого отображения информации. Ими были получены результаты моделирования трех различных программ, написанных на языке ассемблера для кэш-памятей емкостью от 64 до 512 слов.

Они исследовали также три различные стратегии обратной записи: 1) обязательная запись слова и в кэш, и в основную память; 2) запись нового слова сначала в кэш с обязательной записью в основную память, когда слово следует заменить (простой свопинг); 3) использование контрольного бита для указания того, было ли слово модифицировано или нет, и осуществления записи на основании флажка — контрольного бита (флаговый свопинг). Затем был исследован эффект разделения кэша на отдельные буферы для команд и данных. При неразделенном кэше они полагали кэш организованным на пословную выдачу, а разделенный кэш считали организованным как на двухсловную, так и однословную выдачу информации. Было обнаружено, что во всех случаях неразделенная кэш-память обеспечивает лучшую производительность.

Пытаясь доказать техническую целесообразность использования кэш-памяти для малых мини-ЭВМ, они продемонстрировали возможность получения пятикратного выигрыша в производительности при двукратном или меньшем увеличении стоимости.

В 1974 г. Белл и др. [B4] привели дальнейшие результаты моделирования, полученные на мини-ЭВМ PDP-8E. Они детально исследовали три разных метода записи: сквозную запись, конфликтное использование обратной записи (КИЗ) и конфликтное использование записи с битом модификации (КИМ). В бите модификации хранилась информация о том, было ли слово модифицировано или нет. Если слово не модифицировалось, то данные из кэш-памяти во время перемещения информации не перезаписывались в основную память. Было найдено, что метод КИМ дает значительное увеличение производительности по сравнению с методом простой сквозной записи (рис. 7.20).

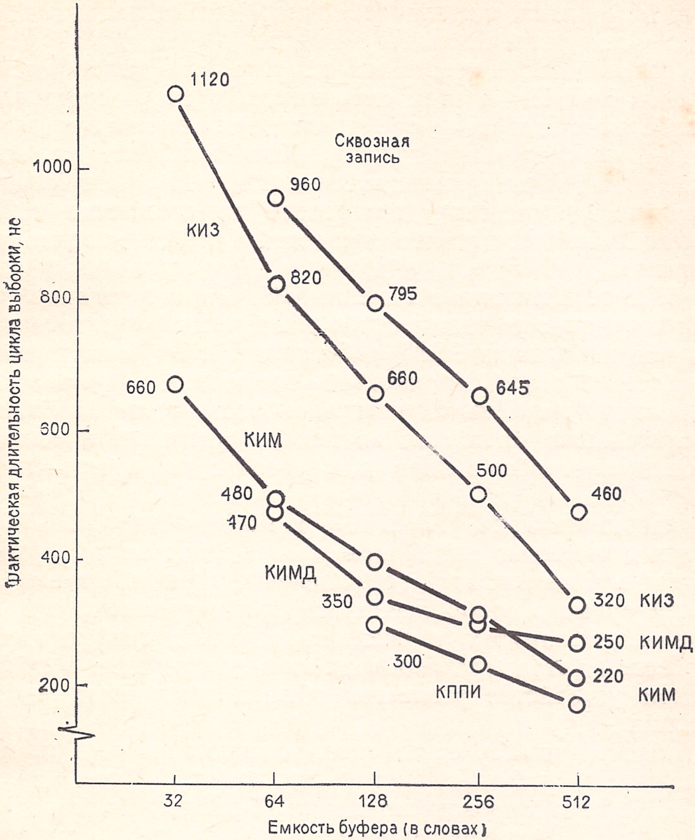


Рис. 7.20. Время выборки данных.

Чтобы собрать информацию, были выбраны три различные программы, дающие информацию каждая о миллионе адресов. Одна программа была программой числовых вычислений, написанная на широко используемом языке FOCAL — диалоговом интерпретаторе. Вторая программа представляла собой алгоритм FFT, являющийся наиболее часто запрашиваемой программой из библиотеки программ. Третья программа была написана на языке ассемблера. Они исследовали также возможность организации кэш-памяти как двух наборов однородных ячеек — команд и данных.

Для разделенной кэш-памяти наилучшая производительность была получена при разделении ее на две равные части — для команд и для данных. Однако достигнутая производительность все же уступала производительности,

получаемой при неразделенной однородной кэш-памяти. Так же как и для универсальных ЭВМ, они исследовали возможность использования в кэше предварительного просмотра информации (КППИ). В этом случае каждое слово поступало из основной памяти, в то же время последующее слово из памяти вводилось в следующую ячейку. Хотя казалось, что такая схема дает лучшую производительность, чем схема без предварительного просмотра, вследствие ее сложности она не представляется достаточно полезной, чтобы вводить ее в кэш-память.

В 1975 г. Пюкнелл в Англии [P5, P6] исследовал улучшение соотношения стоимости и фактической производительности для небольших ЭВМ, работающих с окружающим оборудованием в реальном масштабе времени. Он пришел к выводу, что из всех методов, которые использовались в больших вычислительных системах, концепция кэш-памяти является наиболее многообещающей для увеличения производительности.

Триденник и Уэлч [T7] в 1977 г. исследовали схемы буферизации с помощью кэшей при работе с операндами различной длины. Они показали, что введение трех небольших кэш-памятей прямого отображения информации между медленно действующей памятью и последовательным арифметическим устройством устранило необходимость применения в арифметическом устройстве регистров общего назначения переменной длины.

В области коммерческих мини-компьютеров первым мини-компьютером, использующим кэш-память, была система ЭКЛИПС, высокопроизводительная 16-разрядная машина, использующая микропрограммное управление, арифметику с плавающей запятой и МОП-память с исправлением ошибок емкостью до 256 Кбайт. ЭКЛИПС, имеющая такое оборудование, была спроектирована как высокопроизводительная машина для работы с языками высокого уровня и операционной системой [D2, D3].

Быстродействующая биполярная кэш-память с удобной адресацией емкостью 16 слов была встроена в каждое устройство памяти емкостью 8К слов. Такой подход к использованию распределенной буферной памяти дал возможность процессору получать два байта данных каждые 200 нс и достигнуть общей скорости передачи данных в системе 10^6 байт в секунду. Так как каждое устройство памяти имело свой кэш, то отношение емкостей основной памяти и кэш-памяти оставалось постоянным независимо от того, сколько устройств памяти использовалось. Это позволяло легко расширять память. При запросе со стороны процессора сначала просматривается содержимое памяти адресов для проверки на при-

сутствие данных в кэш-памяти. Если данные находятся в кэше, то в течение 200 нс они поступают в процессор; в противном случае данные выбираются из МОП-памяти за 700 нс.

В кэше емкостью 16 слов могут находиться четыре блока данных, по четыре слова в каждом блоке. В устройстве управления кэшем используется LRU-алгоритм для перемещения любого конкретного блока из кэша. Этот алгоритм непрерывно присваивает каждому блоку в кэше метки, определяющие последний и предпоследний блоки, к которым обращался процессор, а также первый и второй блоки, к которым процессор не обращался дольше всего, и отмечает флагом неиспользуемые данные, делая их ближайшими кандидатами на перемещение информации из кэша.

Далее память распределялась по двум, четырем или восьми магистралям для сокращения фактического времени цикла работы памяти. Четырехкратное расслоение ферритовой памяти на сердечниках дало возможность уменьшения эффективной длительности цикла до 600 нс (во всяком случае, меньше 800 нс), а при использовании восьмикратного расслоения время выборки ферритовой памяти может быть уменьшено до 500 нс. В машине ЭКЛИПС была сделана попытка ввести новые идеи в проектирование системы, выполняемой на старой технологической базе [W7].

Устройство управления памятью и кэш-память машины PDP-11/70

ЭВМ PDP-11/70 была представлена фирмой DEC в 1976 г. в качестве наиболее мощной машины семейства 16-разрядных машин PDP. Эта ЭВМ была спроектирована для использования в очень больших сложных и высокопроизводительных системах — от быстродействующих систем, работающих в реальном масштабе времени, и до многопрограммных систем, работающих в режиме разделения времени [P9]. При расширении физического адресного пространства до 2 Мбайт эта ЭВМ включала три отдельных адресных пространства: виртуальное 16-разрядное программное адресное пространство, 18-разрядную общую шину и 22-разрядное адресное пространство физического адреса (рис. 7.21). 22-разрядное физическое адресное пространство позволяет процессору запрашивать данные из памяти уникального объема в два мегабайта. В этой машине пространство физических адресов (22 разряда) было намного больше, чем ее область виртуальных адресов (16 разрядов). Устройство управления памятью (УУП), встроенное в процессор, преобразовывало 16-разрядный программный виртуальный адрес в 22-разрядный

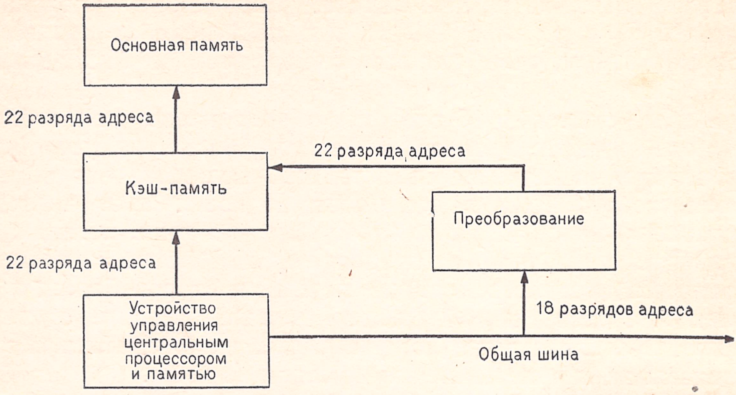


Рис. 7.21. Адресное пространство в ЭВМ PDP-11/70.

физический адрес, а для преобразования 18-разрядного адреса в 22-разрядный физический адрес было введено отдельное устройство отображения общей шины.

Устройство управления памятью

Устройство управления памятью, встроенное в процессор, давало возможность оборудованию осуществлять управление памятью и защиту памяти. В функции управления памятью входят распределение и перераспределение памяти для различных пользователей в многопользовательской мультипрограммной системе, а также контроль за разрешением допуска пользователя к памяти для защиты программы пользователя от злонамеренного использования памяти.

УУП машины PDP-11/70 состояло из трех групп 32 16-разрядных регистров (всего 96 16-разрядных регистров): для внутреннего режима, для режима супервизора и для пользовательского режима. Код режима вводился в слово состояния процессора, которое определяло группу используемых регистров. Каждая группа выделенных регистров далее разделялась на две подгруппы — для области команд и для области данных. Каждая из этих подгрупп далее подразделялась на две части по восемь регистров. Одна часть носила название — адресный регистр страницы (АРС), а другая — регистр дескриптора страницы (РДС), причем АРС и РДС выбирались парой. И адресное пространство, и пространство данных имели по восемь АРС и РДС для каждого режима работы процессора — внутреннего, супервизора и пользовательского (рис. 7.22). Пространство команд использовалось для выборки команд, индексных слов, абсолютных адресов и непо-

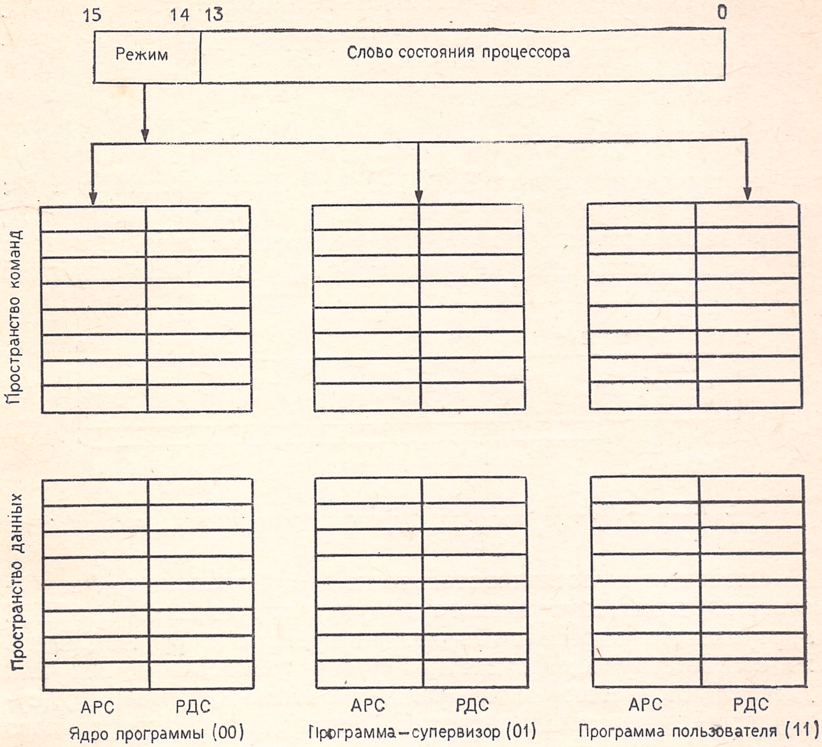


Рис. 7.22. Устройство управления памятью ЭВМ PDP-11/70.

средственных операндов; пространство данных применялось для выборки всей остальной информации.

Пара АРС/РДС содержала всю информацию, необходимую для нахождения и выборки страницы, находящейся в текущий момент в работе. В машине PDP-11/70 память динамически распределялась по страницам, объем памяти в которых колебался от 32 слов до 4К слов (от 64 байт до 8 Кбайт) и которые могли быть расположены где угодно в физической памяти. Каждая страница включала от одного до 128 внутренних блоков длиной по 32 слова; начальный физический адрес каждой страницы представлял собой целое число, умноженное на 32 слова (64 байт), т. е. страницы распределялись в границах по 64 байт. Адресный регистр страницы содержал начальный адрес страницы в качестве номера блока физической памяти, а регистр дескриптора страницы — информацию относительно управления выборкой значения длины страницы и расширения объема страницы. Такая схема работы позволяет динамически перераспределять страницы

в физической памяти совершенно очевидным для программиста образом, просто изменяя содержимое адресных регистров страницы.

Формирование адресов

Виртуальный адрес в ЭВМ PDP-11/70 содержит три поля: сдвоенное трехразрядное поле активной страницы (ПАС), с помощью которого выбирается конкретная пара регистров АРС/РДС, семиразрядный номер блока текущей страницы и шестиразрядное смещение внутри блока (рис. 7.23).

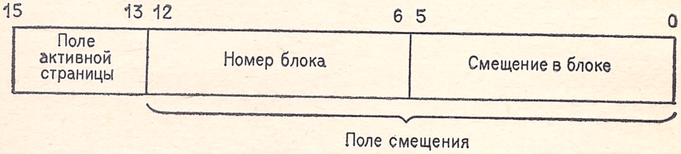


Рис. 7.23. Виртуальный адрес в ЭВМ PDP-11/70 (16 разрядов).

Логическая последовательность формирования физического адреса из виртуального следующая: трехразрядное ПАС виртуального адреса выбирает конкретную пару АРС/РДС из соответствующей области адресов (области команд или данных) соответствующего режима работы (внутреннего, супервизора, пользовательского). Номер блока виртуального адреса сравнивается с полем, указывающим длину страницы (разряды 8—14 ПДС), для выявления любой возможной ошибки в определении границы или длины страницы. Ошибка может возникнуть, во-первых, при увеличении номера блока, когда выделенный номер блока виртуального адреса больше, чем указанный в поле длины страницы (ПДС), и, во-вторых, при уменьшении номера, когда номер блока меньше, чем указанный в поле длины страницы. Если ошибки нет, то номер блока виртуального адреса (ВА) прибавляется к адресному полю страницы (из АРС), чтобы сформировать номер блока, который будет содержать физический адрес блока. Смещение внутри блока относительно поля ВА приставляется к физическому номеру блока, чтобы выявить истинный 22-разрядный физический адрес (рис. 7.24).

Кроме поля длины страницы ПДС содержит также несколько контрольных и информирующих о выборке разрядов. Имеется также разряд, определяющий направление расширения страницы в программной или стековой области. Чтобы обеспечить эффективность схемы управления памятью, в системе используются четыре отдельных регистра управления памятью или регистры исправления ошибок.

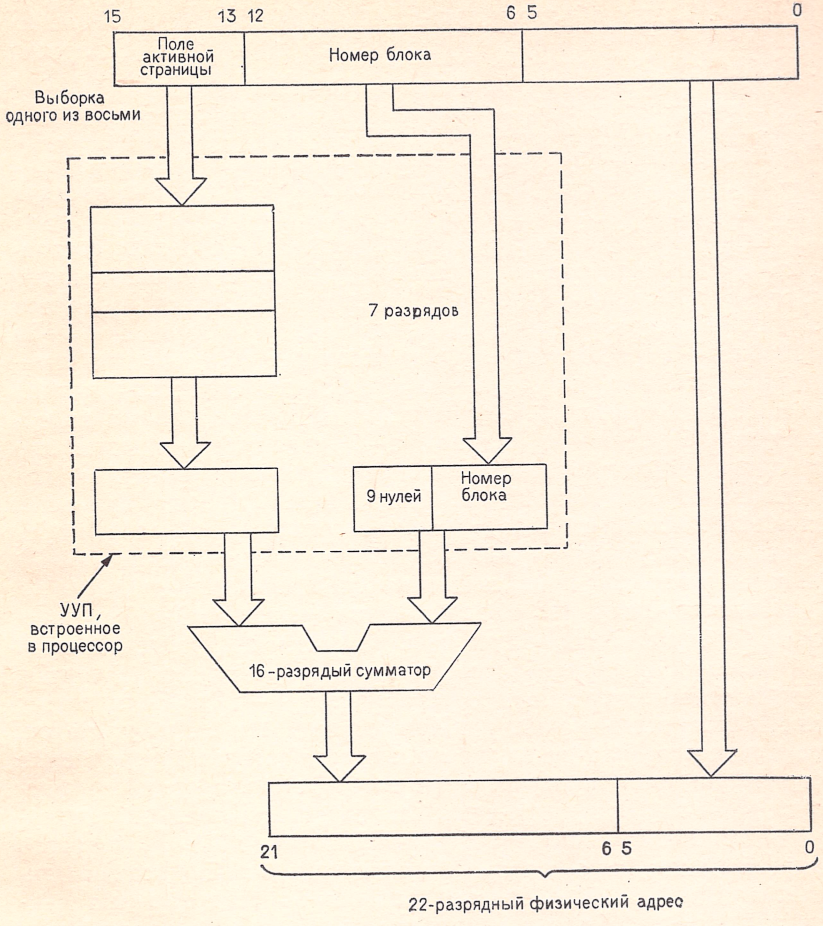


Рис. 7.24. Преобразование виртуального адреса в физический в ЭВМ PDP-11/70.

Кэш данных

Кроме мощного устройства управления памятью PDP-11/70 снабжена также быстродействующей кэш-памятью для данных емкостью 2 Кбайт, обеспечивающей более быструю выборку наиболее часто требуемых данных. После выполнения значительного объема моделирования и макетирования емкость кэш-памяти была выбрана равной 1К слов. Кэш-память организована в виде двух ассоциативных кэшей. Основной единицей передачи данных между основной памятью и кэшем данных явился блок, состоящий из двух слов (четырех

байтов). Таким образом, кэш-память для данных состояла из 512 блоков данных, причем каждый их двух кэшей содержал 256 блоков.

Физический адрес (ФА) в ЭВМ PDP-11/70 состоит из трех полей: нулевой разряд определяет байт внутри слова, первый разряд определяет слово внутри блока, а разряды 2—21 определяют адрес блока. Адрес блока состоит из двух полей: поля индексов и поля меток.

Кэш-память данных организована в виде системы, состоящей из 256 классов и двух ассоциативных кэшей: кэша 0 и кэша 1. Разряды 2—9 (восемь битов) физического адреса являются адресом индекса (класса), а разряды 10—21 (12 бит) — адресом метки. Функционально каждый кэш кэш-памяти состоит из двух частей: массива адресов (меток) и массива данных. И массив меток, и массив данных содержат по 256 величин. Разряды 2—9 являются индексом для массива меток. Однако, поскольку каждая величина для массива данных состоит из блока данных или из двух слов, каждый кэш массива данных имеет емкость 512 слов и требует девять разрядов для адресации слова. Разряды 1—9 являются адресом индекса для массива данных.

Поле индексов для массива меток (разряды 2—9) определяет ячейку в обоих кэшах массива индексов. После выборки метки из обоих кэшей ассоциативно сравниваются с частью метки физического адреса (разряды 10—21). Если имеет место соответствие между полем метки ФА и одной из меток, то считается, что имел место запрос данных и требуемое слово или блок поступают из соответствующего кэша массива данных. Эти физические данные поступают по магистрали шириной 16 бит. Часть физического байта физического адреса может быть связана с индексной частью массива данных для выборки любого конкретного байта требуемых данных. Кроме того, так как кэш-память имеется и для данных, и для команд, то данные, выбираемые из кэш-памяти, могут быть либо физическими данными, запрошенными процессором, либо командой.

Однако если соответствия между полем меток физического адреса и каким-либо значением метки (массива адресов) нет, то считается, что запрашиваемых данных в кэш-памяти нет и требуемый блок данных необходимо получить из основной памяти. Формируется запрос на чтение необходимого блока из основной памяти. Блок отправляется в процессор, запоминается в кэш-памяти, и обновляется соответствующее сообщение в массиве индексов. Если в обоих кэшах значения индексов достоверны для соответствующего класса, то необходимо принять какое-то решение относительно перемещения информации в один из кэшей. В кэш-памяти используется

случайное перемещение, а не алгоритмы LRU или FIFO, которые являются стратегией перемещения при обновлении содержимого кэшей.

В кэшах используется стратегия сквозной записи. При выборке ячейки для записи обновляется содержимое и кэш-памяти, и основной памяти. Однако если в кэше запрашиваемая информация отсутствует, то выбираемая ячейка обновляется только в основной памяти, кэш при этом не изменяется. Так как частота записи составляет обычно 10 % по сравнению с 90 % частоты чтения, то, по-видимому, такая схема сквозной записи не должна налагать слишком высокие требования на систему в целом. Но поскольку основная память обновляется всегда (из-за стратегии сквозной записи), то в системе в любой момент имеются верные данные и нет необходимости передавать какие-либо данные из кэша в основную память. Кроме того, если внезапно пропадет напряжение питания и данные в кэше исказятся, никаких трудностей не возникнет, так как в ферритовой памяти всегда имеется верная информация.

Кроме того, ассоциативный кэш данных в ЭВМ PDP-11/70 обеспечивает возможность работы в усеченном режиме, если в кэше обнаружены какие-либо неисправности. Если один из двух кэшей функционирует неправильно, система может отправлять все запросы в другой кэш. Система обладает также способностью вообще обходить кэш-память и направлять все запросы в основную память, что сопровождается некоторым ухудшением производительности. Все эти возможности реализуются с помощью программируемого управляющего регистра. Следует также отметить, что переход с режима работы с двумя кэшами на режим работы с одним кэшем не обязательно ухудшает производительность на 50 % ввиду статистического характера вероятности удачного обращения к памяти.

PDP-11/60

Позднее кэш-память использовалась также в ЭВМ PDP-11/60 — мини-ЭВМ среднего класса [M7]. Результаты моделирования, полученные Штрекером, которые весьма сильно повлияли на проектирование PDP-11/70, оказали влияние и на PDP-11/60. Однако в этом случае было принято несколько компромиссных решений, которые благоприятно сказались на конструкции и назначении этой машины.

Во-первых, объем блока для PDP-11/60 был выбран равным одному слову вместо двух, так как в качестве магистрали памяти применялась 16-разрядная общая шина. Хотя использование блока объемом в одно слово немного ухудшило

вероятность удачного обращения к памяти (87 % по сравнению с 92 % для PDP-11/70), считалось, что эта машина полностью удовлетворяет поставленным перед ней задачам. Кроме того, с целью достижения приемлемых стоимости и производительности при использовании существующей технологии производства памяти кэш-память была организована в виде системы прямого отображения, а не в виде нескольких кэшей или ассоциативной памяти. Ввиду того что в PDP-11/60 использовался размер блока в одно слово, очень простой стала стратегия перемещения информации. (Фактически не требовалось никакой стратегии перемещения). С другой стороны, в PDP-11/70 использовалась стратегия случайного перемещения.

Чтобы сократить потери времени, связанные с синхронизацией и передачей по общей шине, и для облегчения привязки подсистем существующей памяти к общей шине, кэш-память располагалась между процессором и общей шиной, а не между общей шиной и основной памятью.

Использование кристаллов памяти с бóльшей плотностью и некоторое ухудшение производительности позволили значительно уменьшить число компонент в кэш-памяти. Для кэш-памяти в PDP-11/70 требовалось 440 микросхем, тогда как для кэша в PDP-11/60 — всего 85 микросхем. Этого удалось добиться за счет более простой организации кэша и использования кристаллов памяти с бóльшей плотностью информации.

Так как 16-разрядные мини-ЭВМ заменялись 32-разрядными, стало очевидным, что организация кэш-памяти будет иметь существенное значение для обеспечения соответствующего быстродействия памяти высокопроизводительных мини-ЭВМ. Двумя примерами таких машин являются PDP VAX-11/780 и машина фирмы Data General MV/8000.

Система VAX-11/780

VAX-11/780 — это 32-разрядная высокопроизводительная мини-ЭВМ, разработанная фирмой DEC в 1978 г. Эта система служила расширителем виртуальных адресов (VAX) семейства PDP-11 и обеспечила более 4 млрд. байт адресного пространства (32 разряда), отражаемых на более чем 1 млрд. байт физического адресного пространства (30 разрядов). Преобразование виртуальных адресов в физические осуществлялось с помощью микропрограмм, система имела три механизма ускорения преобразования — командный буфер, буфер преобразования адресов (БПА) и кэш-память для данных, — которые обеспечивали высокую производительность. 8-байтовый командный буфер обеспечивал достаточный

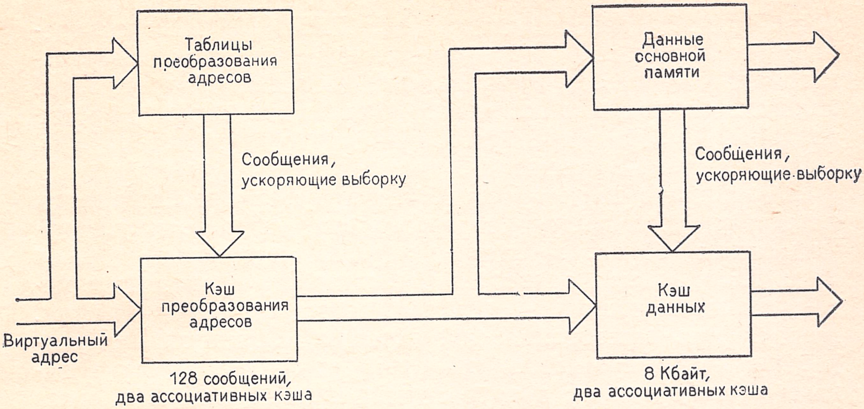


Рис. 7.25. Буфер преобразования адресов и кэш данных для ЭВМ VAX-11/780.

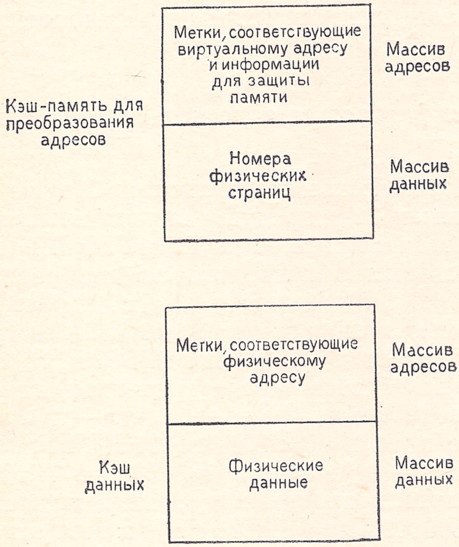


Рис. 7.26. Кэш для преобразования адресов и кэш данных для ЭВМ VAX-11/780.

для процессора предварительный просмотр команд и давал процессору возможность выбирать и декодировать следующую макрокоманду, когда заканчивалось выполнение текущей команды. Буфер преобразования адресов объемом 128 ячеек служил кэшем для виртуальной адресной и ассоциативно защищенной информации, а 8-килобайтовый двойной ассоциативный кэш обеспечивал быстродействующую выборку

часто запрашиваемых данных (рис. 7.25) [D4]. Буфер преобразования адресов при поступлении на его вход виртуального адреса формирует на выходе физический адрес, тогда как кэш данных принимает этот физический адрес (сформированный буфером преобразования адресов) и передает на выход требуемые данные.

Принципиально (и фактически) и буфер преобразования адресов и кэш данных содержат два раздельных информационных объекта: массив адресов и массив данных (рис. 7.26). Массив адресов является просто хранилищем меток, а массив данных — памятью соответствующих данных. Для буфера преобразования адресов массив адресов содержит метки, соответствующие виртуальному адресу и защитной информации; в случае кэша данных массив адресов содержит метки, связанные с физическим адресом. Массив данных буфера преобразования адресов содержит физический адрес (физическую страницу с номером), тогда как массив данных кэша содержит фактические физические данные.

Виртуальное и физическое адресные пространства VAX-11/780

Виртуальное адресное пространство объемом более 4 млрд. байт определяется 32-разрядным виртуальным адресом и находится в адресах с нулевого по FFFFFFFF. Все виртуальное адресное пространство разделено на две области: область, предназначенную для вычислений (с адреса 00000000 по 7FFFFFFF, объем свыше 2 млрд. байт), и область системных программ (с адреса 8000000 по FFFFFFFF, объем свыше 2 млрд. байт). Область, предназначенная для вычислений, подразделяется далее на две подобласти — программную подобласть P0 (с адреса 00000000 по 3FFFFFFF) и управляющую подобласть P1 (с адреса 4000000 по 7FFFFFFF). Четверть виртуального пространства, находящаяся в старших адресах (с C0000000 по FFFFFFFF), в этот момент не определена [D4].

Все виртуальное адресное пространство разделено на страницы по 512 байт, причем границы этих страниц программистам не известны. Виртуальный адрес, формируемый процессором, определяет: 1) является ли область адресов вычислительной или системной, 2) номер виртуальной страницы и 3) физическое смещение внутри страницы. Соответствие номеров виртуальной страницы номеру физической страницы в основной памяти хранится в основной памяти в виде страничных таблиц. Кроме информации, предназначенной для выполнения преобразования адресов, каждое сообщение стра-

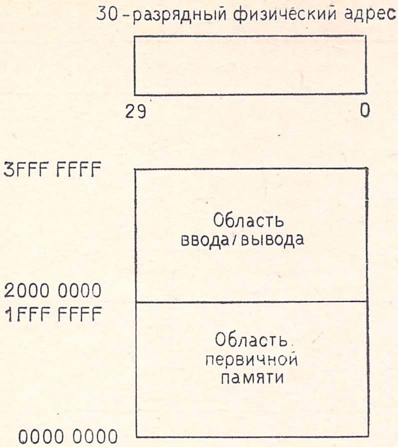


Рис. 7.27. Пространство физических адресов.

ничной таблицы содержит некоторую информацию для защиты соответствующей виртуальной страницы. Так как таблицы отображения страниц могут быть очень большими, буфер преобразования адресов служит существенным ускорителем при получении сообщений из страничных таблиц.

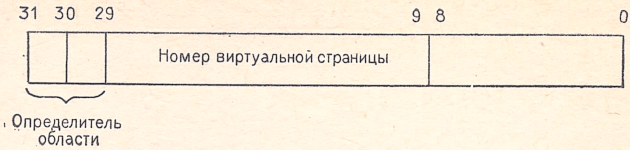
Физический адрес в машине VAX-11/780 занимает 30 разрядов (объем более 1 млрд. байт). Физическое адресное пространство состоит из двух областей: области первичной памяти и области ввода/вывода (рис. 7.27).

Архитектура виртуальной и физической областей памяти оказывает определенное влияние на организацию преобразования в буфере и кэше данных. Чтобы обеспечить быструю выборку информации как из системного, так и вычислительного пространства адресов, буфер преобразования адресов содержит сообщения для системной и вычислительной областей адресов.

Разделение буфера преобразования адресов на системную и вычислительную области дает возможность селективного аннулирования записей только в страничной таблице вычислительного пространства при свопинге. Так как системная область используется при всех вычислениях, эти сообщения не являются неработающими во время свопинга, что улучшает таким образом общую производительность системы.

Преобразование адресов

Как показано на рис. 7.28а и 7.28б, виртуальный адрес (ВА) в системе VAX-11/780 состоит из трех полей: смещения байта внутри страницы, индексной части и некоторой инфор-



Разряды 31	30	Область
1	0	Область программ
0	1	Область управления
1	0	Системная область
1	1	Резерв

Рис. 7.28а. Формат виртуального адреса ЭВМ VAX-11/780.

мации в виде меток. Разряды с 0 до 8 ВА определяют смещение байта внутри страницы, разряды с 9 по 13 и 31 разряд ВА, разделяющий системную и вычислительную области, служат индексом, а разряды 14—30 виртуального адреса содержат номер виртуальной страницы. Шесть разрядов индекса (разряды 31 и 13—9) служат адресом индекса или адресом класса для массива адресов и массива данных буфера преобразования адресов. И массив адресов, и массив данных буфера преобразования адресов организованы в виде двух ассоциативных кэшей. Каждый кэш массива адресов содержит 32 сообщения для системной области и 32 сообщения для вычислительной области, всего в одном кэше 64 сообщения.

Поле индексов виртуального адреса выбирает ячейку в обоих кэшах массива данных и массива адресов. После выборки метки из обоих кэшей массива адресов ассоциативно сравниваются с меткой виртуального адреса. Если имеет место соответствие между полем метки виртуального адреса и меткой одного из кэшей, то считается, что был запрошен адрес, и из соответствующего массива данных выбирается соответствующий номер блока данных физической страницы. Этот номер блока данных физической страницы занимает 21 разряд. Смещение байта относительно поля виртуальных адресов приставляется к номеру физической страницы, чтобы сформировать 30-разрядный физический адрес. Кроме того, во время преобразования адресов проверяется код защиты виртуального адреса на возможность нарушения запрета выборки. Если такое нарушение имело место, то вызывается программа обработки такого нарушения (рис. 7.29).

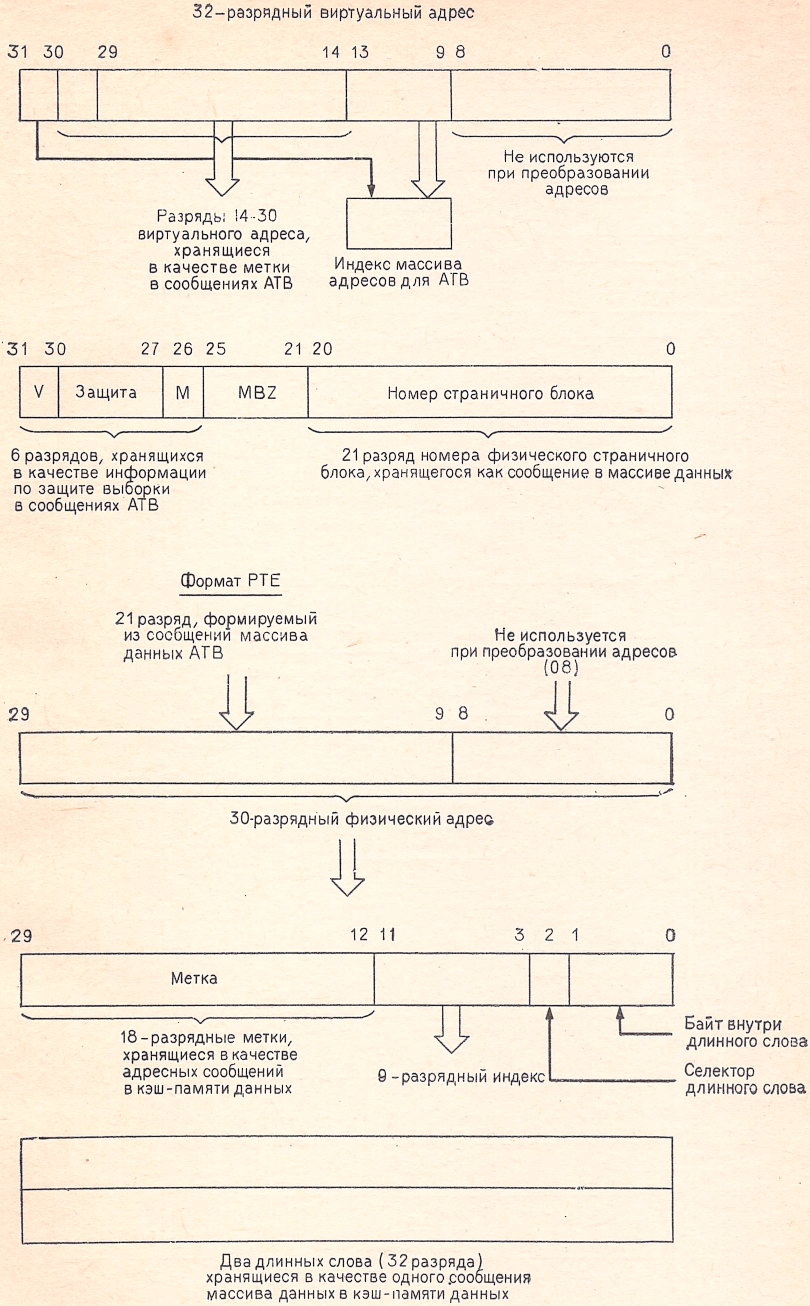


Рис. 7.28б. Формат виртуального адреса, формат PTE, индексирование кэша данных и сообщение кэша. (ATB — address translation buffer, буфер преобразования адресов, БПА.)

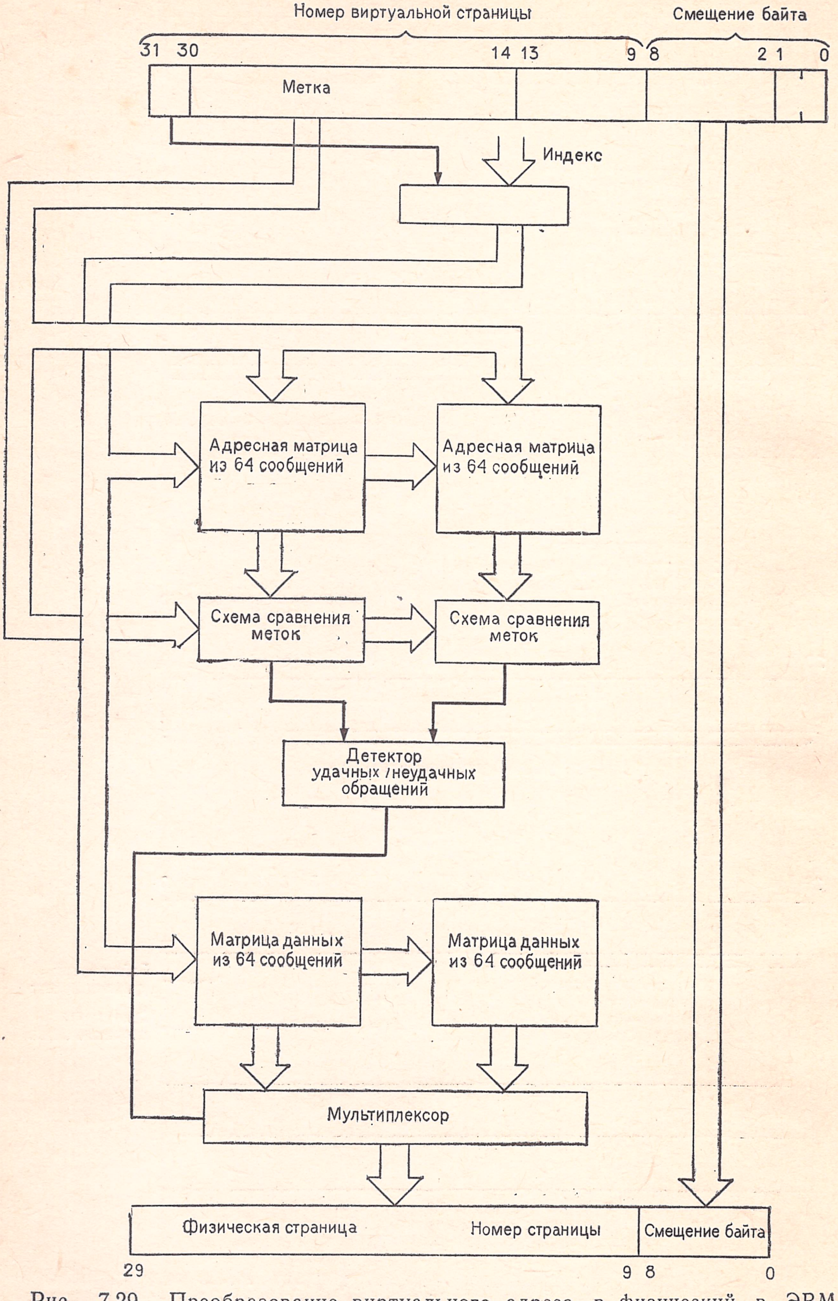


Рис. 7.29. Преобразование виртуального адреса в физический в ЭВМ VAX-11/780.

Однако если соответствия между полем метки виртуального адреса и любой из меток массива адреса нет, то считается, что запрос был ошибочным и формируется системное микропрерывание. Микропрерывание вызывает микропрограмму, которая пытается обновить запись, выданную в кэш преобразования адресов и полученную из соответствующего сообщения PTE (из основной памяти). Если микропрограмме удастся получить сообщение из PTE (заметьте, что она может выдать ошибку страницы в зависимости от того, какое сообщение поступило из вычислительной таблицы страниц), то она пытается скорректировать массив адресов и массив данных буфера преобразования адресов и организовать вызов повторно. Если соответствующий разряд достоверности установлен (это означает, что вызывается верная страница), следующая попытка будет удачной. Если бит достоверности сброшен, то вызывается системная программа, которая, отвечая за расположение соответствующей страницы, вводит ее, если необходимо, в основную память, корректирует PTE и затем пытается повторить вызов.

Так как буфер преобразования адресов может вместить максимум 128 сообщений, то в каждый данный момент он может ускорить обработку максимум 128 разных виртуальных страниц. Кроме того, буфер преобразования адресов формирует 30-разрядный физический адрес, который должен пройти через следующий уровень преобразования прежде, чем будут получены действительные данные. Это отображение обеспечивается кэшем данных.

Кэш данных

Так же как буфер преобразования адресов (БПА), кэш-память данных организована в виде двух ассоциативных кэшей — кэша 0 и кэша 1. Каждый кэш, также как и БПА, разделен на две части — массив меток адресов и массив данных. Так как счетверенное слово (два длинных слова, или четыре слова, или восемь байт) является основной единицей обмена информацией между кэш-памятью и основной памятью, то каждому значению массива адресов (кэша данных) соответствует одно сообщение матрицы данных, состоящее из одного счетверенного слова. Сообщения массива адресов обеспечивают отображение физического адреса (формируемого БПА) в физический адрес кэш-памяти, а сообщения массива данных обеспечивают выдачу фактических физических данных.

Для достижения наибольшей вероятности удачного обращения при заданных соотношениях между стоимостью и технологической базой кэш-память данных (и массив адресов,

и массив данных) организована в виде двух ассоциативных кэшей по 512 сообщений в каждом. Эти 512 сообщений могут рассматриваться как 512 различных классов. Так как каждое сообщение из массива данных состоит из одного счетверенного слова, то емкость каждого кэша массива данных составляет 512 счетверенных слов (или 4 Кбайт), а общая емкость кэш-памяти данных составляет 1К счетверенных слов, или 8 Кбайт.

Работа кэш-памяти данных

Физический адрес, формируемый БПА, состоит из трех полей: разряды 0—2 определяют адрес длинного слова и смещение байта внутри этого слова, разряды 3—11 определяют адрес индекса, или класса, а разряды 12—29 — метку физического адреса. Отметим, что массив адресов кэш-памяти данных не имеет защиты информации и не разделяется на системные или физические адресные данные. Разделение на системную и вычислительную области памяти может быть проведено только для виртуальных адресов, а не для данных.

Кроме того, отметим, что хотя каждый кэш массива данных кэш-памяти данных содержит 512 сообщений, каждое из сообщений состоит из счетверенного слова, или двух длинных слов. Поэтому каждый кэш массива данных в действительности состоит из 1024 длинных слов. Итак, индексный адрес для массива адресов (512 входных сообщений) и массива данных (1024 входных сообщения) имеет разную длину, и должен существовать механизм адресации к ним. В то время как разряды физического адреса (ФА) 3—11 служат адресом индекса массива адресов, разряд 2 ФА (определяет длинное слово) совместно с разрядами 3—11 ФА является адресом индекса для массива данных.

Поле индексов ФА (разряды 3—11) массива адресов определяет ячейку в обоих кэшах массива адресов. Метки, выбранные из обоих кэшей массива адресов, ассоциативно сравниваются с меткой ФА. Если имеет место соответствие между полем метки ФА и меткой любого кэша, то вызов считается вызовом данных и соответствующее счетверенное слово выбирается из соответствующего кэша массива данных (рис. 7.30). Эти физические данные имеют магистраль шириной 32 разряда. Смещение физического байта физического адреса может быть присоединено к выходной информации массива адресов для выборки конкретного байта требуемых данных. Отметим также, что эти данные могут быть либо запрошены процессором, либо даже быть командой, которую можно загрузить в буфер команд. Однако если соответствия между полем меток ФА и любой меткой массива адресов

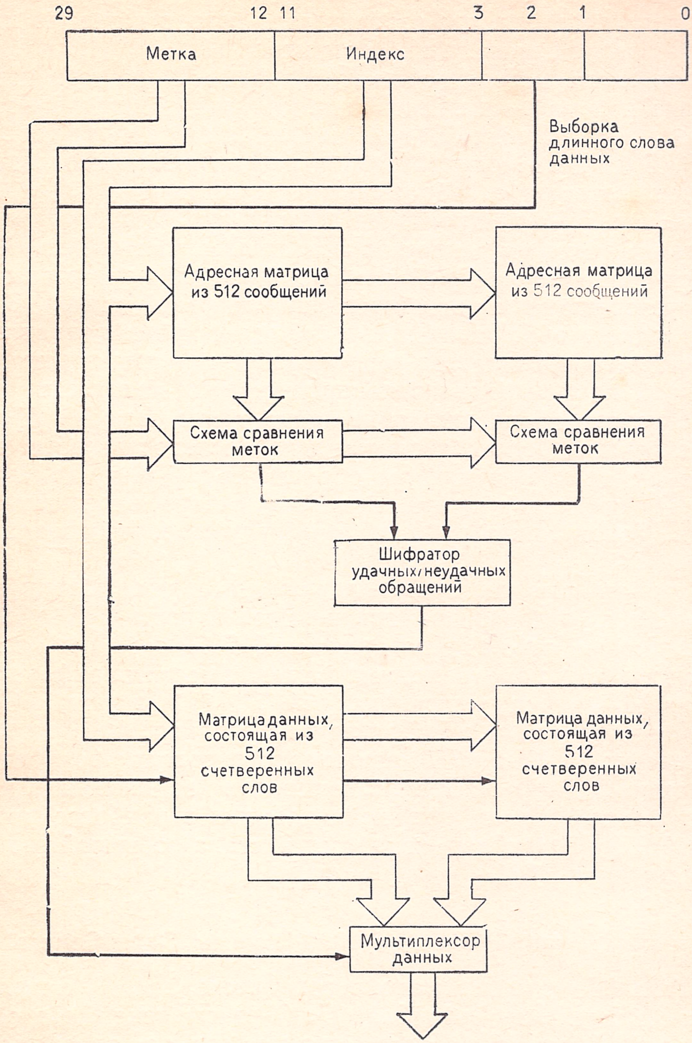


Рис. 7.30. Кэш-память данных для ЭВМ VAX-11/780.

нет, то считается, что не было ответа на запрос данных и формируется микропрерывание для обработки этого случая.

В системе VAX-11/780 для кэш-памяти данных различаются случаи отсутствия ответа на запрос чтения и запрос записи. Если произошла потеря запроса чтения, то счетверенное слово (два длинных слова) должно быть полностью восстановлено выборкой его из основной памяти с записью в соответствующий кэш. Если доступны оба кэша, то следует при-

нять некоторое решение относительно того, в каком из кэшей следует корректировать информацию.

В качестве метода перемещения информации при корректировке кэша в кэш-памяти данных принята стратегия случайного перемещения без использования алгоритмов LRU или FIFO. В качестве случайного бита фактически используется триггер, который перебрасывается в каждом цикле работы до тех пор, пока не произойдет потеря выборки. В случае отсутствия ответа на запрос записи вызываемая ячейка обновляется только в основной памяти. В кэш-памяти данных используется схема модифицированной сквозной записи. В случае удачного обращения к кэш-памяти с целью записи в ней информации обновляется информация и в кэш-памяти, и в основной памяти. Однако на процессор не налагается никаких ограничений на ожидание конца цикла записи информации в память. Только в случае двух последовательных записей или в случае неудачного обращения к буферу команд, за которым следовала запись, процессор ставится в режим ожидания. Такая схема работы также улучшает производительность.

Отметим, что кэш-память данных ускоряет выборку длинных слов из основной памяти и имеет емкость 8 Кбайт. Так как в каждой странице содержится 512 байт данных, то теоретически кэш-память данных может вместить информацию либо о 16 различных страницах (по 512 байт в каждой), либо 64 байт информации с 512 различных страниц.

Система VAX-11/780 продемонстрировала возможность полномасштабного внедрения концепции кэш-памяти для преобразования адресов и хранения данных в наиболее мощные мини-ЭВМ. С тех пор кэш-память стала обычной во всех мощных мини-ЭВМ.

Кэш-память мини-ЭВМ MV/8000 фирмы Data General

MV/8000 — это высокопроизводительная 32-разрядная мини-ЭВМ фирмы Data General. О ее выпуске было объявлено в 1980 г. [D5]. Предусмотрена полная аппаратная и программная совместимость этой ЭВМ со всем набором 16-разрядных вычислительных систем, не имеющих встроенных средств выполнения сложных арифметических операций (семейство машин ЭКЛИПС). В архитектуре MV/8000 использованы виртуальное адресное пространство объемом 4,3 млрд. байт (32 разряда) и физическое адресное пространство объемом 512 Мбайт. Первая реализация ЭВМ с такой архитектурой могла работать с пространством физических адресов объемом 2 Мбайт. Преобразование виртуального адреса в физический выполняется микропрограммно, система снабжена

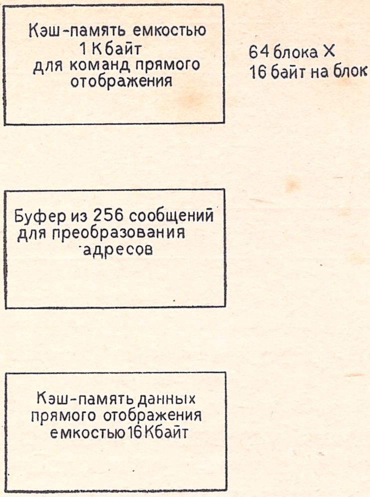


Рис. 7.31. Механизмы ускорения выборки информации для MV/8000.

тремя элементами ускорения такого преобразования: кэш-памятью команд, буфером преобразования адресов и кэш-памятью данных, что позволяет достигнуть высокой производительности. Кэш-память команд емкостью 1 Кбайт (64 блока по 16 байт) с непосредственным отображением информации обладает достаточными возможностями по просмотру содержимого соседних ячеек памяти с целью обеспечения конвейерной реализации работы процессора с высокой производительностью. Буфер преобразования адресов объемом 256 слов служит кэшем для виртуальной адресной информации и защищает адресную информацию, а кэш-память прямого отображения емкостью 16 Кбайт (1К блоков по 16 байт каждый) обеспечивает ускоренную выборку наиболее часто запрашиваемых данных (рис. 7.31). Буфер преобразования адреса на входе имеет виртуальный адрес, а на выходе — физический адрес, тогда как кэш данных получает физический адрес (формируемый буфером преобразования адресов) и выдает требуемые данные.

Операционная система AOS/VS мини-ЭВМ MV/8000 разделяет 4,3 млрд. байт виртуального адресного пространства на две особые области: область программ пользователя и область системных программ. Программы пользователя занимают нижнюю половину виртуального адресного пространства (до 2,15 млрд. байт), а системные программы — остальную половину адресного пространства. Эти области не перекрываются.

9 Зак. 99

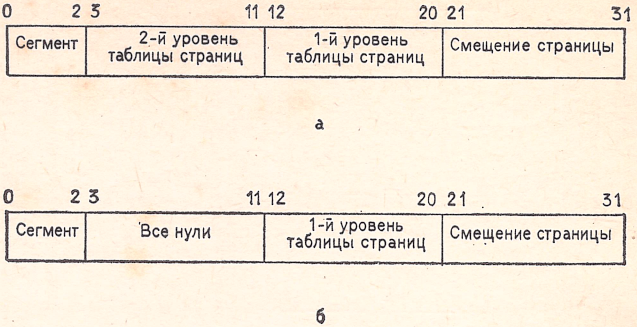


Рис. 7.32. Одно- (*б*) и двухуровневый (*а*) форматы виртуального адреса в MV/8000.

Для эффективного управления таким огромным виртуальным адресным пространством полное виртуальное адресное пространство разделено на восемь меньших виртуальных областей, называемых сегментами. Преобразование виртуального адреса в физический выполняется на основе сегментов с использованием одно- или двухуровневой структуры построения таблицы страниц. Логическое и физическое адресные пространства разделены на блоки по 2Кбайт в каждом, а оборудование обеспечивает непосредственную работу со страничной структурой, содержащей по 2 Кбайт в странице (рис. 7.32).

Так как логическое адресное пространство намного больше, чем физическое адресное пространство, то все страницы информации с диска или какой-либо дополнительной памяти не могут поместиться в основной памяти. Поэтому в системе для отображения адресов логических страниц на физические применена схема преобразования, использующая последовательность страничных таблиц. Это преобразование выполняется устройством преобразования адресов (УПА). Сообщение страничной таблицы выдает информацию о состоянии страницы, свободна ли она, может ли быть выбрана и находится ли в настоящий момент в оперативной памяти. Кроме того, с каждой страницей связана некоторая информация о защите выборки: можно ли ее использовать только для чтения или только для записи и т. д. Эти страничные таблицы хранятся в основной и вспомогательной памяти. На рис. 7.33 и 7.34 показан механизм формирования адреса для системы MV/8000.

Для ускорения выборки физических страниц (и для того, чтобы избежать выборки сообщений из страничной таблицы) формирование физических адресов выполняет аппаратное УПА, содержащее до 256 адресов различных логических стра-

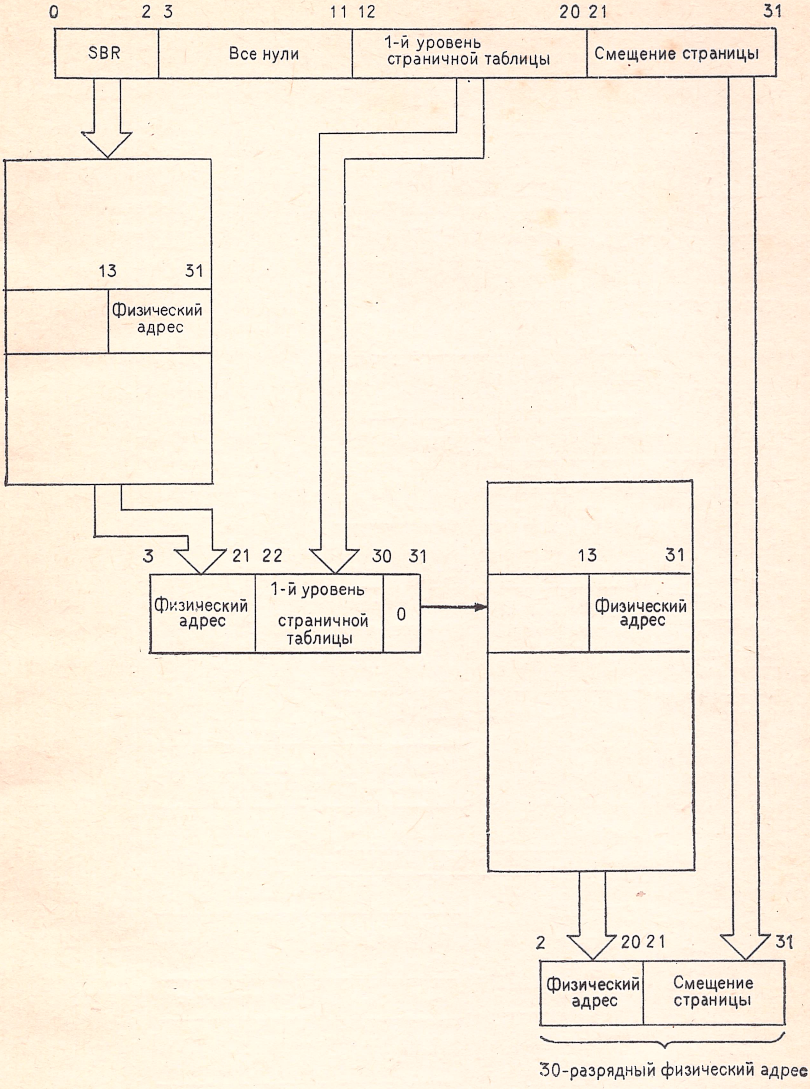


Рис. 7.33. Одноуровневое преобразование адресов с помощью страничной таблицы в MV/8000.

ниц, которые могут чаще всего запрашиваться процессором.

Кроме сообщений страничной таблицы, ускоряющих работу системы, сообщения УПА содержат два дополнительных разряда информации, помогающей операционной системе более эффективно управлять работой со страницами, когда

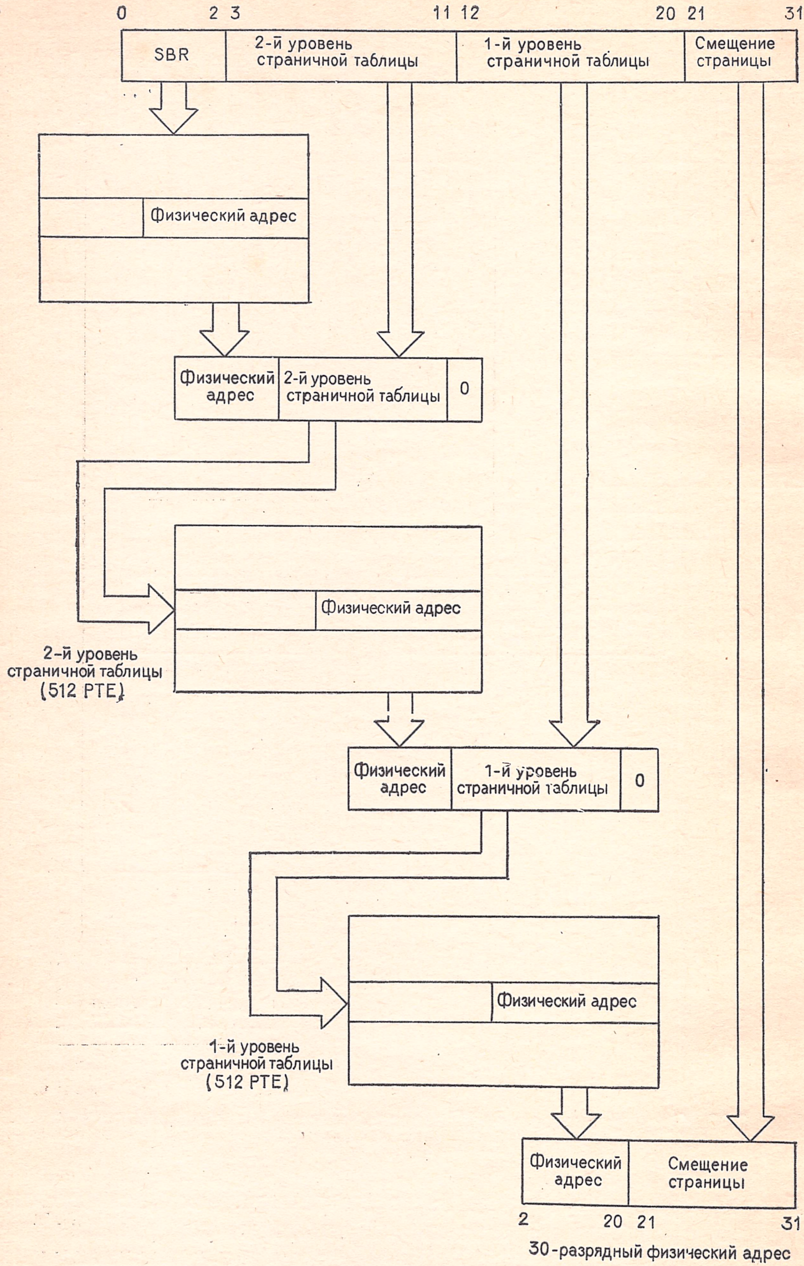


Рис. 7.34. Двухуровневое преобразование адресов с помощью страничной таблицы в MV/8000.

имеет место неудачное обращение при выборе страницы. Этими двумя разрядами являются бит модификации и бит выборки. Неудачное обращение при выборке страницы происходит, когда осуществляется обращение к логической странице, которой в настоящее время нет в оперативной памяти. Каждый раз, когда фиксируется неудачное обращение при вызове страницы, необходимо переписать затребованную страницу из вспомогательной памяти в оперативную память. В зависимости от свободного места, имеющегося в оперативной памяти, может появиться необходимость вывести страницу из оперативной памяти, чтобы освободить место для новой страницы. Операционная система использует бит модификации информации, чтобы определить, может ли быть передана конкретная страница, выбранная для вывода из памяти. Если бит модификации равен нулю, т. е. страница не изменилась с тех пор, как ее последний раз ввели с диска, а на диске уже имеется достоверная информация, то нет необходимости снова переписывать эту страницу на диск. Однако если бит модификации равен единице, значит, страница модифицировалась после поступления ее в память и ее необходимо переписать на диск.

Бит выборки информации также используется операционной системой, чтобы определить частоту выборки отдельных страниц. Это помогает операционной системе заменить наиболее редко вызываемую страницу.

Кроме того, система MV/8000 использует механизм свопинга *затребованной страницы* для улучшения использования основной памяти. Такая процедура использования затребованных страниц означает, что логические страницы хранятся на диске, а не в основной памяти и вызываются в основную память только по требованию. Механизмом листания страниц управляет программа обработки прерывания, возникающего из-за отсутствия страницы. При появлении такого прерывания включается программа обработки этого прерывания, которая формирует запрос соответствующей страницы, передает страницу в память и обновляет данные в соответствующей страничной таблице.

Буфер преобразования адресов содержит сообщения о 256 страницах и организован в форме кэша с прямым отображением и объемом в 256 слов. Кроме буфера преобразования адресов система MV/8000 снабжена также кэш-памятью данных.

Кэш-память данных

Кэш-память данных (называемая также системным кэшем) мини-ЭВМ MV/8000 — это кэш-память емкостью

16 Кбайт с циклом выборки четырех байт, равным 110 нс. Кэш спроектирован таким образом, что процессор может запрашивать данные каждый машинный цикл. Процессор выбирает из кэша четыре байта данных за 220 нс, а восемь байт — за 330 нс.

И основная память, и системный кэш данных разделены на блоки по 16 байт, причем такой блок является основной единицей обмена информацией между основной памятью и кэшем. Кэш-память представляет собой кэш прямого отображения, имеющий 1 К 16-байтовых блоков. Любой блок системного кэша содержит 16 последовательных байтов информации из основной памяти. Однако, поскольку использован кэш прямого отображения, существует взаимно-однозначное соответствие между блоками основной памяти и блоками системного кэша. Хотя архитектура MV/8000 и позволяет работать с 512 Мбайт физического адресного пространства, первая реализация MV/8000 могла работать лишь с 2 Мбайт физического адресного пространства. Для этой реализации основную память можно представить разделенной на 128 сегментов, каждый из которых состоит из 1К блоков. Системный кэш данных имеет емкость максимум от одного до трех сегментов. Однако, поскольку кэш организован на основе блоков, определенный блок конкретного сегмента может оставаться в фиксированном месте кэша. Например, нулевой блок сегментов 1, 2, 3 и т. д. может быть на месте нулевого блока системного кэша и т. д. Такая схема прямого отображения исключает ассоциативный поиск для большого числа блоков, и отпадает необходимость в большом числе компараторов и мультиплексоров.

Ввиду того что первая реализация системы работала с физическим адресным пространством объемом в 2 Мбайт, только разряды 1—31 (младшие 31 разряд) физического адреса используются кэшем в качестве исполнительного адреса. Этот исполнительный адрес состоит из двух полей: разряды 28—31 (четыре бита) определяют смещение байта внутри блока; разряды 11—27 определяют адрес блока. Фактически адрес блока в свою очередь состоит из двух полей: поля индексов и поля меток. Так как кэш имеет емкость 1К и использует прямое отображение, то разряды 18—27 (10 разрядов) служат полем индексов, а разряды 11—17 (семь разрядов) — полем меток. Поскольку кэш состоит из блоков, метка соответствует блоку.

Кэш-память данных можно рассматривать как память, состоящую из двух массивов: массива меток и массива данных. Поле индексов исполнительного адреса (разряды 18—27) определяет местоположение конкретного блока массива меток и массива данных. Будучи выбранной, метка из массива ме-

ток сравнивается с полем меток исполнительного адреса (разряды 11—17). Если они равны, то считается, что имеет место запрос данных, и смещение байта фактического адреса присоединяется к полю индексов для выборки соответствующих данных из кэша в процессор.

Если же метка запрашиваемого блока не найдена в массиве меток, то соответствующий блок данных необходимо выбрать из основной памяти, послать в процессор и запомнить в кэше на случай выборки этого блока в будущем. Конкретное слово, запрашиваемое процессором, является первым словом, выбираемым из основной памяти, и оно первым засылается в процессор. Затем из блока последовательно выбирается остальная информация.

Так как кэш-память организована на основе прямого отображения, то и схема перемещения работает соответствующим образом. Существует только одно место, где может остаться запрашиваемый блок.

В системном кэше при записи используется стратегия обратной записи. Поэтому в нем имеется бит модификации, который содержит информацию о том, необходимо ли блок снова записывать в основную память. При командах записи обновляется информация только в блоке кэш-памяти. После извлечения блока, если его содержимое модифицировалось, он сначала переписывается в основную память, а затем информация в блоке обновляется в соответствии с изменениями.

7.5. УПРАВЛЕНИЕ ЗАПОМИНАЮЩИМИ УСТРОЙСТВАМИ И РАБОТА С КЭШ-ПАМЯТЬЮ В МИКРОПРОЦЕССОРАХ

При постоянном развитии технологической базы и быстром уменьшении стоимости оборудования начали производиться микропроцессоры, обладающие возможностями мощных мини-ЭВМ. Созданы четыре поколения микропроцессоров от простейших четырехразрядных микропроцессоров до наиболее сложных 32-разрядных процессоров. Производительность и возможности первых двух поколений микропроцессоров весьма ограничены. Адресное пространство не превышало 64 Кбайт, программное обеспечение выполнялось на языках ассемблера и управление памятью не являлось сложной задачей. Однако возможности адресации памяти для микроЭВМ третьего и четвертого поколений значительно возросли. Емкость адресуемой памяти для некоторых микроЭВМ третьего поколения составляет: 1 Мбайт для Intel 8086, 16 Мбайт для Motorola 68000 и 16- и 32-разрядных микроЭВМ фирмы National, 8 Мбайт для изделий фирмы Zilog и микроЭВМ AMD28000. Емкость адресуемой памяти для не-

которых микроЭВМ четвертого поколения, таких, как IAPX432 фирмы Intel, составляет 2^{40} байт. (Это намного больше, чем адресное пространство большинства универсальных и мощных 32-разрядных мини-ЭВМ.)

При таких больших емкостях адресуемой памяти методы управления памятью, т. е. проблемы эффективного распределения и перераспределения памяти с целью оптимизации ее полного использования, а также защиты содержимого памяти от несанкционированного доступа были распространены и на микроЭВМ. Подход к решению задачи управления памятью, применяемый к основной памяти, был предложен и для микроЭВМ.

Устройство управления 16-разрядной памятью

Необходимость обеспечить возможность работы с языками высокого уровня, сложными операционными системами, большими программами и базами данных, а также уменьшить стоимость запоминающих устройств ускорили проникновение в микроЭВМ запоминающих устройств большой емкости. Управление виртуальной памятью, которое использовалось в универсальных и мини-ЭВМ, стало обычным в высокопроизводительных микроЭВМ. Обычные методы использования виртуальной памяти, такие, как сегментация, страничная организация, сегментация со страничной организацией, стали применяться и в микроЭВМ. В области 16-разрядных микроЭВМ разработчики виртуальной памяти разделились на два лагеря. Фирмы AMD, Zilog и Intel придерживались схем сегментации памяти, а фирмы Motorola и National внедряли схемы линейной памяти.

УУП микро ЭВМ 8010 фирмы Zilog

Микросхемы Z8010 фирмы Zilog и AMZ8010 фирмы AMD (второй производитель) являются устройствами управления памятью первого поколения для 16-разрядных микропроцессоров. Эта микросхема обеспечивает сегментное управление виртуальной памятью, подобное схемам управления, разработанным для различных универсальных и мини-ЭВМ.

Пространство логических адресов центрального процессора Z8001 разделено на множество линейных адресных областей, каждая из которых, имеющая определенную длину или размер, называется сегментом. Любая величина, находящаяся в сегменте, выбирается с помощью двухэлементного адреса: первый элемент определяет номер сегмента (селектор сегмента), а второй — смещение выбираемой величины относительно базы сегмента (селектор перемещения).

Центральный процессор Z8001 обеспечивает работу сегментного адресного пространства объемом 128 логических сегментов, каждый из которых содержит 16 Кбайт, в результате чего формируется адресное пространство объемом 8 Мбайт. Центральный процессор Z8001 формирует логический адрес, который состоит из 7-разрядного номера сегмента и 16-разрядного смещения (рис. 7.35). Кроме того, процес-

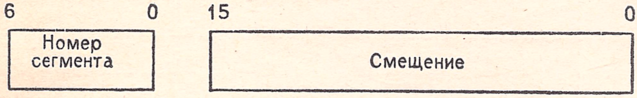


Рис. 7.35. 23-разрядный логический адрес микропроцессора Z8001.

сор формирует также сигналы состояния, которые показывают, осуществляет ли он выборку команды, выборку данных или операцию над областью стека и используется ли адресное пространство пользователем или супервизором. Итак, теоретически общее логическое пространство адресов этого процессора составляет 48 Мбайт. Эти 48 Мбайт пространства

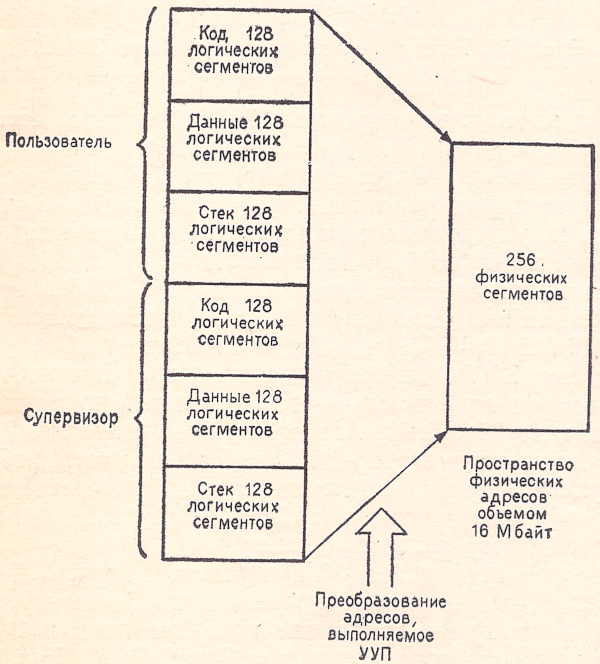


Рис. 7.36. Отображение логического адресного пространства в физическое с помощью микросхемы Z8001.

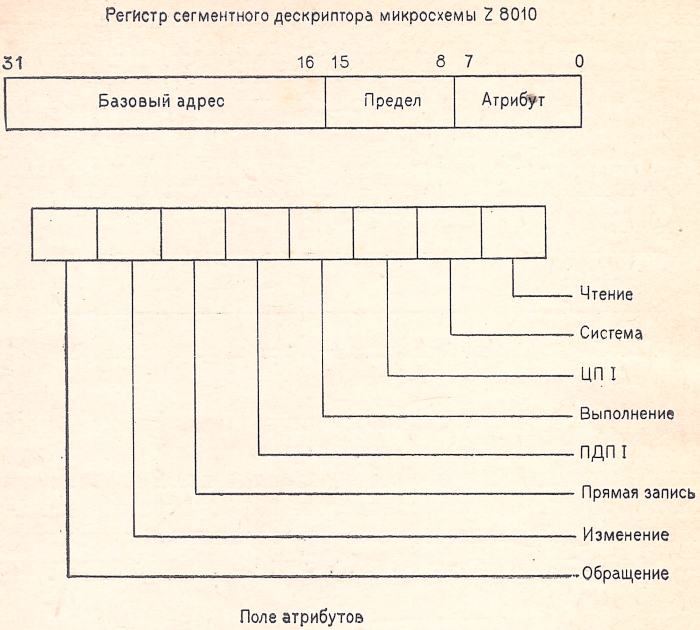


Рис. 7.37. Регистр сегментного дескриптора Z8010.

логических адресов эквивалентны 768 логическим сегментам, каждый из которых имеет объем в 14 Кбайт.

Преобразование логического адреса в 24-разрядный физический адрес осуществляется УУП (рис. 7.36). Одно УПП обеспечивает управление шестьюдесятью четырьмя логическими сегментами по 64 Кбайт в каждом (общее пространство логических адресов составляет 4 Мбайт) и отражает эти логические сегменты на физическое пространство адресов объемом 16 Мбайт (256 физических сегментов).

В системе Z8001 сегменты памяти представляют собой наборы последовательных ячеек, изменяющихся по длине от 256 байт до 64 Кбайт. Адреса сегментов кратны 256 байт (т. е. младшие восемь разрядов адреса базы сегмента всегда равны нулю). Переход от логических к физическим сегментам выполняется с помощью таблицы сегментов, которая содержит дескриптор каждого сегмента.

Дескриптор сегмента в УУП представляет собой 32-разрядное слово, содержащее четыре поля: 16-разрядную базу сегмента, 8-разрядное граничное поле сегмента и 8-разрядное поле принадлежности сегмента (рис. 7.37). Так как восемь младших разрядов адреса базы равны нулю, они не содержатся в дескрипторе сегмента. Поле базы сегмента опре-

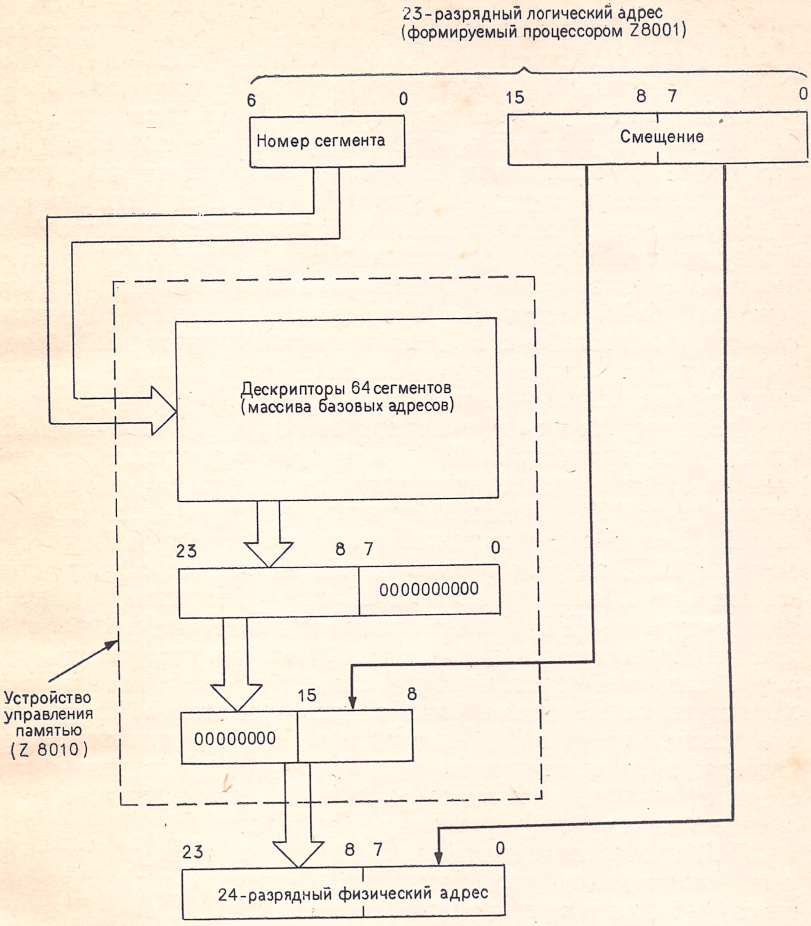


Рис. 7.38. Генерация физического адреса для УУП Z8010.

деляет начальный физический адрес логического сегмента. Граничное поле определяет длину сегмента, измеренную в блоках по 256 байт в каждом. Поле принадлежности содержит восемь разрядов атрибутов сегмента. Так как каждое УУП работает с 64 сегментными дескрипторами, то два УУП должны управлять 128 логическими сегментами, которыми может непосредственно управлять микроЭВМ Z8001. Каждое УУП можно запрограммировать на управление сегментами с 4-го по 63-й или же сегментами с 64-го по 127-й.

Преобразование логического адреса в физический, называемое перераспределением памяти, показано на рис. 7.38. Цифровая часть логического сегмента в логическом адресе

используется в качестве индекса при выборе конкретного дескриптора сегмента в таблице сегментов. Чтобы сформировать 24-разрядный физический адрес, УУП приплюсовывает значащий разряд адреса смещения к 16-разрядному полю базы сегмента соответствующего дескриптора сегмента. Так как восемь младших разрядов базы сегмента всегда равны нулю, то восемь младших разрядов смещения сегмента не участвуют в прибавлении базового адреса к смещению. Вместо этого они просто приписываются к результату в соответствии с алгоритмом формирования 16-разрядного адреса (старший байт смещения к старшим 16 разрядам базового адреса), для того чтобы сформировать 24-разрядный физический адрес. Такое преобразование исключает необходимость применения 24-разрядного сумматора.

Кроме перераспределения адресов другими важными задачами, выполняемыми УУП, являются управление сегментами и защита от несанкционированной или случайной выборки (рис. 7.39). Чтобы обеспечить защиту различных областей памяти от несанкционированного или случайного доступа, в УУП имеется восьмиразрядное поле атрибутов, связанное с каждым сегментом. Процессор обеспечивает выдачу информации в УУП о виде взаимодействия памяти, определяемого словом состояния. Для каждого запроса памяти эти слова состояний сравниваются с атрибутами, связанными с соответствующим сегментом. При несовпадении УУП формирует сигнал прерывания, прерывая работу центрального процессора, и запоминает соответствующую информацию о состоянии, которое может опрашиваться процессором, чтобы определить конкретную причину прерывания.

УУП проверяет каждый запрос памяти по двум параметрам, являющимся условиями прерывания: нарушение процесса выборки и предупреждение о попытке записи. При попытке обращения к памяти в режимах, при которых чтение не разрешено, а разрешено только выполнение операции, при запрете обращения к памяти со стороны процессора, при запрете прямого доступа к памяти, в случае когда для выборки сегмента не хватает атрибутов или при попытке обращения к участку памяти, находящемуся вне определенной длины сегмента, вырабатывается сигнал нарушения процесса выборки. Прерывание, предупреждающее о попытке записи информации в память, вырабатывается УУП в том случае, когда процессор делает попытку записи информации в последние 256 ячеек сегмента, используемого в качестве стека. При формировании такого запроса на прерывание УУП выдает в процессор предупреждение, что стек может переполниться, и вызывает служебную программу операционной системы для обработки этого прерывания. Если требуется, то

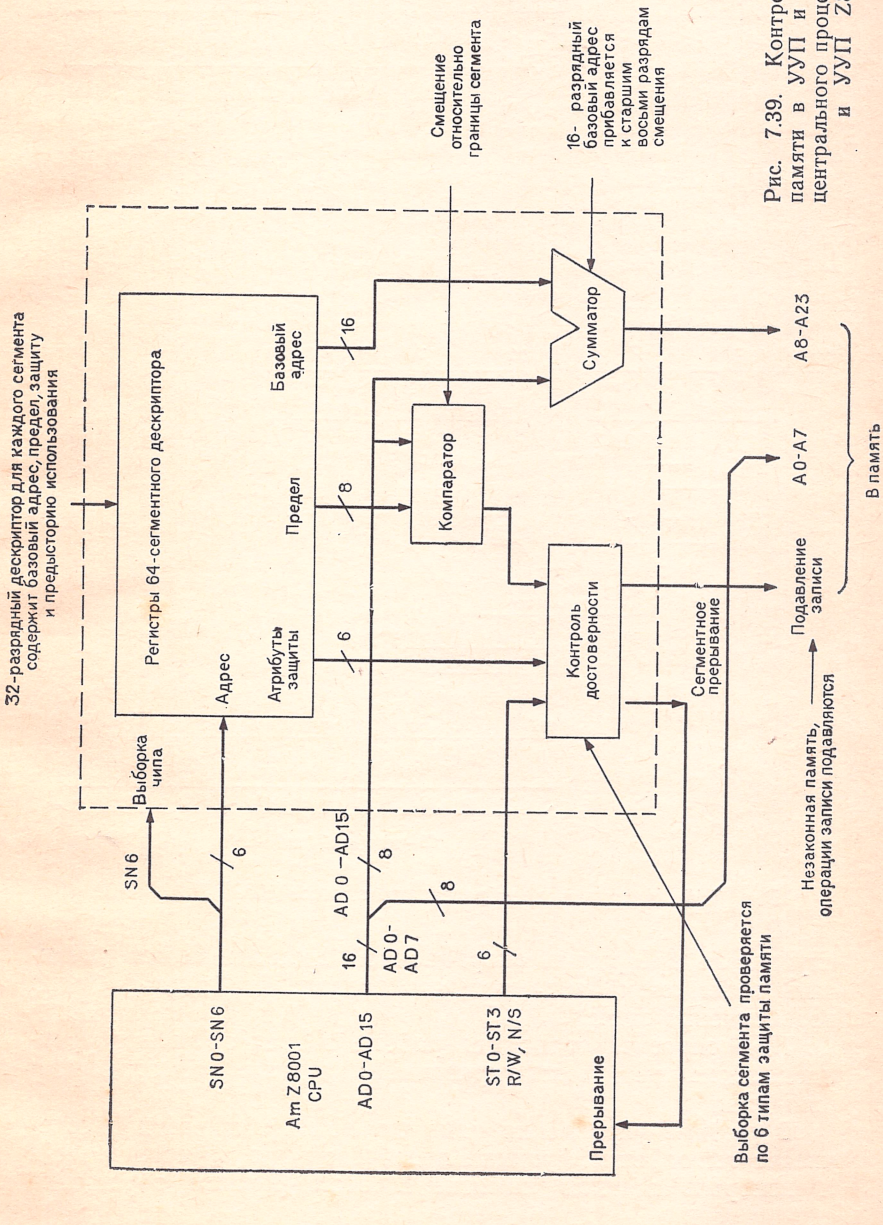


Рис. 7.39. Контроль защиты памяти в УУП и взаимосвязь центрального процессора Z8001 и УУП Z8010.

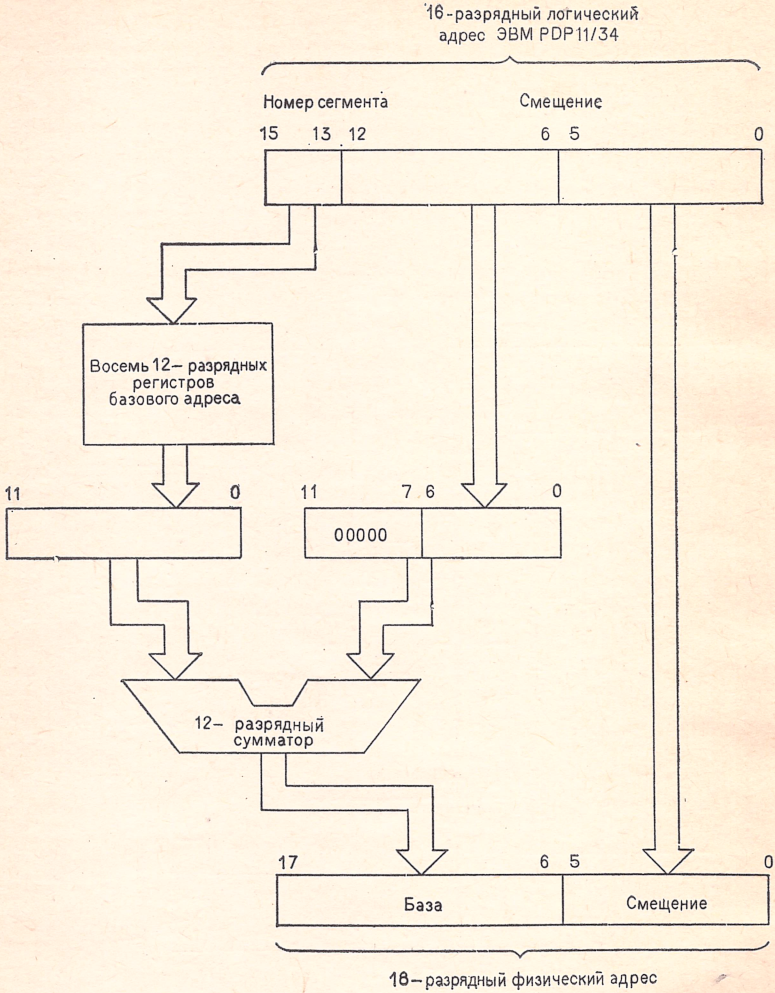


Рис. 7.40. Механизм преобразования адресов ЭВМ PDP 11/34.

служебная программа операционной системы может увеличить объем стека в памяти, что позволит избежать общего переполнения стека.

УУП спроектировано для работы в трех основных функциональных режимах: управления памятью, командном и ожидания.

В режиме управления памятью УУП фактически обеспечивает выполнение функции преобразования адресов памяти и контроль защиты памяти. В этом режиме обращение

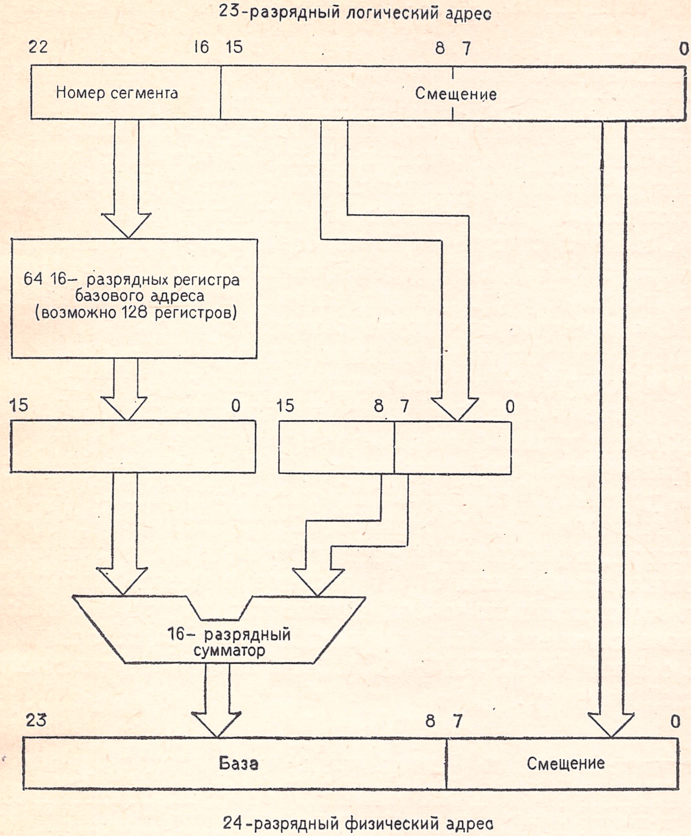


Рис. 7.41. Механизм преобразования адресов устройством управления памятью Z8010.

к памяти со стороны процессора проверяется УУП на разрешение доступа, а затем логический адрес преобразуется в физический. В режиме ожидания УУП отвечает на 22 специальные команды ввода-вывода, поступающие из процессора на загрузку базовых адресов, регистра предельного адреса и регистра атрибутов для дескриптора сегментов — таблицы сегментов. В этом режиме, кроме того, процессор может запрашивать информацию о состоянии УУП. В режиме ожидания выходные адресные шины УУП находятся в третьем высокоимпедансном состоянии. Режим УУП определяется информацией о состоянии, поступающей по шине состояний процессора.

Функция преобразования адресов УУП Z8010 очень похожа на аналогичную функцию мини-ЭВМ PDP 11/34. То, что они похожи, видно из рис. 7.40 и 7.41.

Хотя процессоры Z8001 и УУП Z8010 могут работать с очень большим адресным пространством и обладают прекрасными возможностями по управлению памятью, они не могут работать с виртуальной памятью. Система управления сегментной виртуальной памятью требует оборудования для динамического распределения памяти и свопинга сегментов. Свопинг сегментов включает запоминание текущего состояния процессора (для защиты от ошибки в компоновке сегмента), инициирование выборки соответствующего сегмента, запись информации в требуемый сегмент и запуск процессора с того места, где его работа была прервана. Запоминание состояния процессора должно выполняться самим процессором. При обнаружении ошибки в компоновке сегмента УУП Z8010 формирует сигнал сегментной ошибки; однако процессор Z8001 не может прерывать выполнение текущей выполняемой команды и запоминать свое состояние.

16-разрядный микропроцессор Z8003 второго поколения может управлять виртуальной памятью [C9]. Эта микросхема включает оборудование, обеспечивающее сегментное прерывание, и устраняет необходимость во многих инициализирующих программах. В микросхеме имеется вход прекращения выполнения команды, с помощью которого можно повторить выполнение одной команды процессора. Эта микросхема совместно с УУП Z8010 обеспечивает управление виртуальной памятью.

Управление памятью 8-разрядных микропроцессоров

Концепция управления памятью, впервые использованная в 16-разрядных микропроцессорах, была применена и к 8-разрядным процессорам. Микросхема MC6829 фирмы Motorola является устройством управления памятью для 8-разрядного микропроцессора MC6809. С помощью разделения памяти на страницы фиксированной длины по 2 Кбайт УУП расширяет границы адресации памяти 8-разрядного процессора с 64 Кбайт до 2 Мбайт. Оно использует очень простую схему отображения страниц для преобразования 16-разрядного логического адреса процессора в 21-разрядный физический адрес [R3].

16-разрядный логический адрес процессора состоит из двух полей. Разряды 0—10 (11 бит) определяют смещение байта внутри страницы, а разряды 11—15 (пять бит) — номера логической страницы. Итак, пространство логической адресации состоит из 32 логических страниц, каждая страница объемом по 2 Кбайт. В УУП имеется 128 регистров отображения, разделенных на четыре группы по 32 регистра, один регистр на одно задание. Каждый регистр отображения

является 10-разрядным, чтобы обеспечить хранение старших десяти разрядов физического адреса. Поэтому в любой заданный момент времени задание может получить прямой доступ к 64 Кбайт памяти, находящимся на 32 страницах — по 2 Кбайт на странице. Однако если запросы физической памяти в задании превышают адресное пространство объемом 64 Кбайт, то регистры отображения для соответствующей задачи могут быть перезагружены, чтобы обеспечить большее физическое адресное пространство.

Хотя в микросхеме и имеются четыре группы по 32 регистра, только одна группа регистров действует в одно время. Конкретная группа регистров выбирается операционным ключом, который подключает одну из четырех магистралей. Однако микросхема спроектирована таким образом, что в системе могут находиться восемь таких микросхем, что позволяет обеспечивать работу непосредственно встроенными регистрами 32 групп памяти по 64 Кбайт (максимальный объем 2 Мбайт).

Преобразование адресов

Преобразование логического адреса в физический выполняется непосредственно. Старшие пять разрядов логического адреса (разряды 11—15) выбирают конкретный регистр отображения соответствующего задания (установленного операционным ключом номера задания). Соответствующий регистр отображения формирует старшие десять разрядов физического адреса. Младшие 11 разрядов логического адреса (разряды 0—10) непосредственно добавляются к десяти разрядам физического адреса, сформированным регистром отображения, что создает 21-разрядный физический адрес. Преобразование логического адреса в физический с помощью УУП занимает 110 нс (рис. 7.42).

Микросхема MC6829, однако, не обеспечивает непосредственной защиты страницы и независимый свопинг страниц. При необходимости свопинга страниц программист должен записать команду соответствующей загрузки регистров отображения. Микросхема имеет удобные средства для такого обновления содержимого регистра отображения.

УУП микроЭВМ 16082 фирмы National

Устройство управления памятью 16082 фирмы National Semiconductor является первым УУП, которое обеспечивает работу по запросу со страничной виртуальной памятью [N3, K9, L6]. В этой микросхеме используется сочетание сегментации со страничной организацией для обеспечения работы

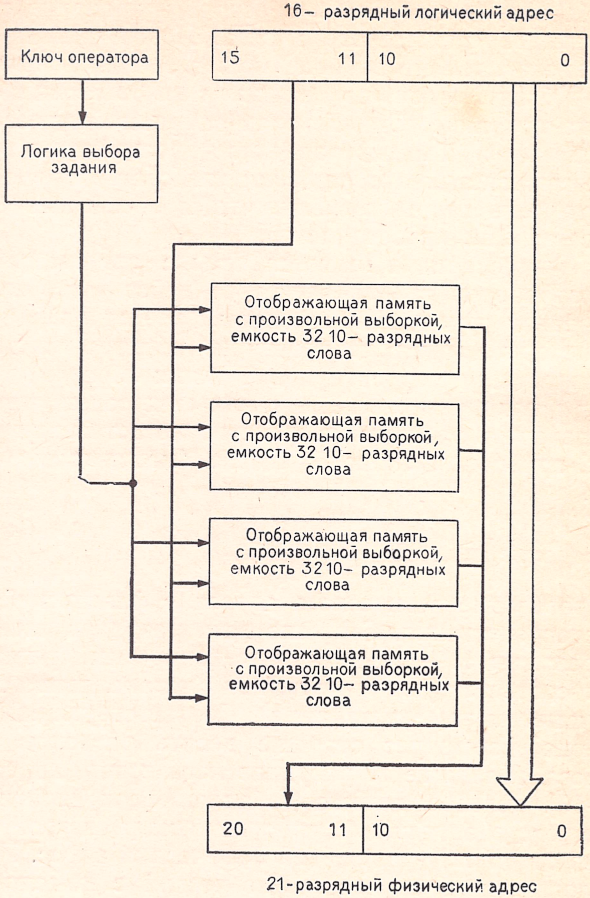


Рис. 7.42. Механизм преобразования адресов в микросхеме MC6829 фирмы Motorola.

с архитектурой виртуальной памяти центрального процессора 16032. Процессор 16032 работает с однородным адресным пространством объемом 16 Мбайт (т. е. и пространство логических адресов, и физическое адресное пространство имеют объем по 16 Мбайт). Хотя оба адресных пространства одинаковы, отображение логического адреса в пространство физических адресов выполняется микросхемой УПП 16082. Кроме логики, обеспечивающей преобразование адресов, микросхема УУП позволяет осуществлять защиту памяти.

Преобразование адресов

И логическое, и физическое пространства памяти процессора 16032 разделены на страницы по 512 байт. Поэтому для управления виртуальной памятью оба адресных пространства разделены на 32 768 страниц, по 512 байт в каждой. Чтобы обеспечить отображение пространства логических адресов в пространство физических адресов, обычная схема использования страничных таблиц потребовала бы очень большого объема страничной таблицы. Для минимизации размера таблицы отображения был использован остроумный двухуровневый подход, заключающийся в комбинировании сообщений из страничной таблицы и таблицы указателей.

Логический адрес, формируемый процессором 16032, состоит из двух полей (как показано на рис. 7.43): разряды

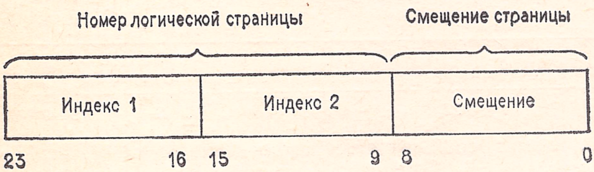


Рис. 7.43. Формат виртуального адреса микросхемы NS16082 фирмы National.

0—8 определяют смещение внутри 512-байтовой страницы, а разряды 9—23 (15 разрядов) определяют адрес логической страницы. Адрес логической страницы делится опять на два поля: входной адрес для 8-разрядной страничной таблицы и входной 7-разрядный адрес таблицы указателей. 8-разрядный входной адрес страничной таблицы определяет адрес индекса для страничной таблицы первого уровня, которая содержит 256 сообщений, каждое из которых имеет длину 32 разряда. В УУП имеется два страничных базовых регистра СБР1 и СБР2: один — используемый пользователем, другой — супервизором, которыми управляет операционная система при изменении значений в страничной таблице.

Старшие 8 разрядов виртуального адреса умножаются на 4 и складываются с соответствующим начальным базовым адресом страничной таблицы, чтобы выделить одно из ее 256 сообщений. Каждое сообщение страничной таблицы содержит 15-разрядный номер блока страницы, с помощью которого выбирается одна из 256 таблиц указателей. Каждая таблица указателей содержит 128 сообщений. Следующие 7 разрядов виртуального адреса (разряды 9—15) умножаются на 4 и складываются с номером блока страницы для выделения одного из 128 сообщений таблицы указателей. Каждое сооб-

щение таблицы указателей содержит 32 разряда; оно формирует 15-разрядный номер блока физических страниц.

Младшие 9 разрядов виртуального адреса присоединяются к этому 15-разрядному номеру блока физических страниц, формируя 24-разрядный физический адрес.

Избыточность такой схемы не очень велика. Объем страничной таблицы составляет 1024 байт, или 2 страницы; каждая таблица указателей имеет объем 512 байт, или одну страницу. Следовательно, для системы требуется две страницы для страничных таблиц и 256 страниц для таблиц указателей. Однако только страницы с содержимым страничных таблиц необходимо постоянно иметь в основной памяти. Страницы с таблицами указателей могут оставаться на диске (вторичной памяти) и вводиться в основную память по требованию. Формат сообщения страничной таблицы и формат сообщения таблицы указателей приведены на рис. 7.44.

Зарезервировано	Номер страничного блока	Зарезервировано	M	R	PL	V
32	23 … 9	8 … 5	4			0

Рис. 7.44. Формат сообщения страничной таблицы (PTE) микросхемы NS16082 фирмы Nationl.

УУП содержит 32 сообщения в буфере преобразования (ассоциативном кэше). Когда в УУП поступает 24-разрядный виртуальный адрес, он ассоциативно сравнивает старшие 15 разрядов виртуального адреса с 32 сообщениями, находящимися в буфере преобразования. Эти сообщения содержат последние выбираемые системой виртуальные адреса со своими преобразованными физическими адресами.

Если ассоциативное сравнение дает положительный результат, то формируется преобразованный адрес, который сразу передается в процессор. Однако если соответствия нет, то УПП обращается к страничной таблице и таблице указателей, находящимся в памяти, и пытается обновить информацию в буфере. УУП использует два базовых регистра страничных таблиц для выбора соответствующих страничных таблиц. Процесс преобразования адресов изображен на рис. 7.45.

Контроль защиты памяти

Кроме преобразования адресов УУП обеспечивает защиту памяти для обеспечения многопрограммного режима работы. Эта проверка защиты обеспечивается с помощью двухуровневых сообщений в страничных таблицах. Страничная таблица с индексом 1 и страничная таблица с индексом 2 содержат соответственно 256 и 128 сообщений.

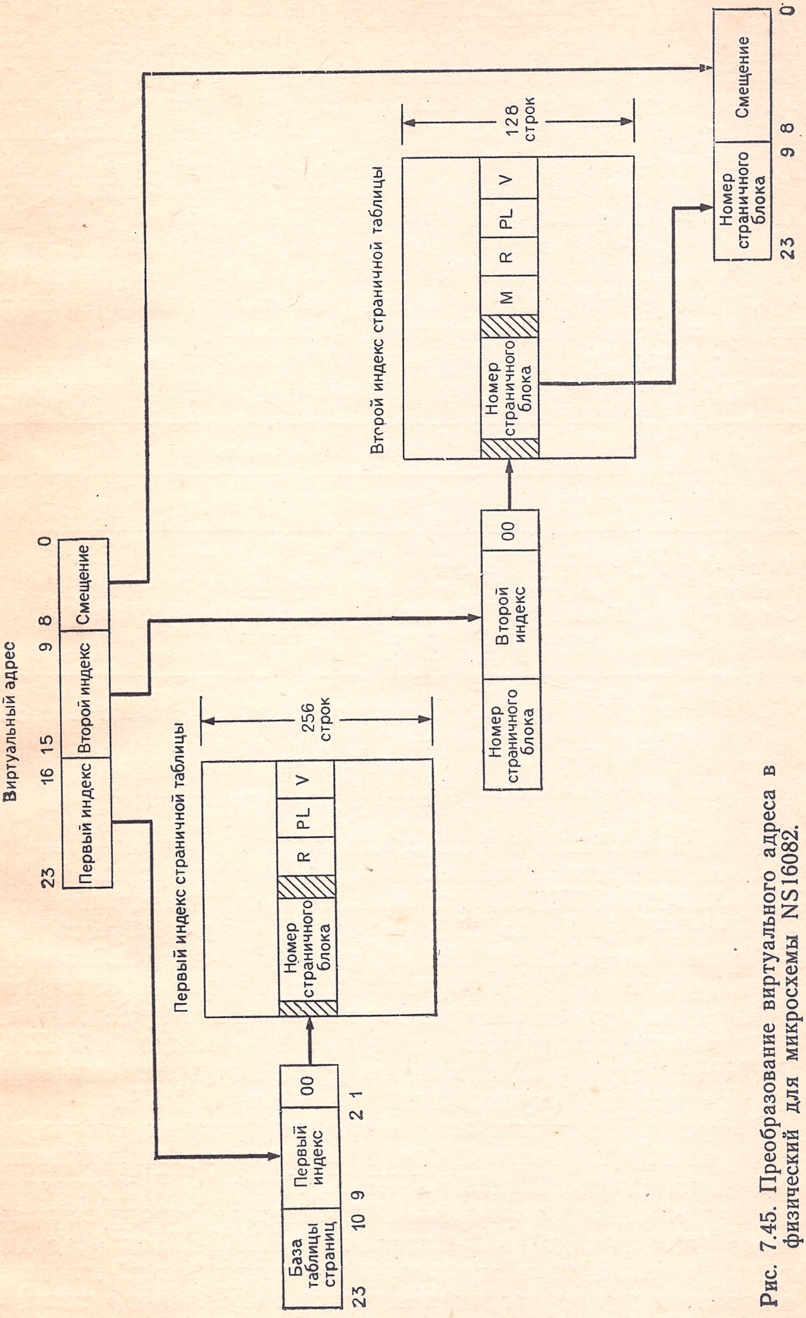


Рис. 7.45. Преобразование виртуального адреса в физический для микросхемы NS16082.

Каждое сообщение страничной таблицы имеет 8 бит информации контроля доступа, из которых в настоящее время используется 5 бит информации, а 3 бит зарезервированы на будущее. Этими битами являются: бит достоверности (V), бит запроса (R), бит модификации (M) и два бита уровня защиты (PL). Значения этих битов следующие.

V бит: этот бит сообщения из страничной таблицы определяет, достоверно ли ассоциированное сообщение.

R бит: бит запроса из сообщения страничной таблицы определяет, запрашивалась ли данная конкретная страница.

M бит: этот бит определяет, модифицировалась ли страница, соответствующая страничной таблице с индексом 2.

PL: биты уровня защиты определяют взаимоотношения между пользователем, супервизором и битом уровня защиты.

Режим	8 бит	Биты уровня защиты			
		00	01	10	11
Пользователь	1	Нет доступа	Нет доступа	Только чтение	Полный доступ
Супервизор	0	Только чтение	Полный доступ	Полный доступ	Полный доступ

Заметим, что бит модификации используется только в сообщениях страничных таблиц с индексом 2.

Обеспечение функционирования виртуальной памяти

Для обеспечения работы с виртуальной памятью УУП соединяется с процессором 16032 двумя отдельными выводами: FLT (СМЕНА ШИНЫ) и прекращение выполнения программы (ЗАПРЕТ). Когда УУП определяет, что требуется выбрать сообщение из страничной таблицы, находящейся в основной памяти, для того чтобы совершить преобразование адресов, оно временно запрещает выполнение текущего цикла выборки памяти путем возбуждения вывода «СМЕНА ШИНЫ». Этот вывод заставляет процессор отключить свою шину, делая ее доступной для использования УУП.

Когда УУП заканчивает выборку информации, оно разрешает закончить текущий цикл выборки путем изменения уровня на выводе СМЕНА ШИНЫ. Когда УУП определяет, что затребованная виртуальная страница не находится в основной памяти, оно возбуждает шину ЗАПРЕТ, которая вынуждает процессор прекратить выполнение текущей команды и организовать переход к программе операционной системы для обработки этого прерывания. Состояние и режим работы

процессора необходимо запомнить, чтобы через некоторое время можно было выполнить прерванную команду снова. Для обеспечения работы виртуальных схем памяти эта возможность выполнить команду повторно чрезвычайно важна, так как команду необходимо повторить после запрета ее выполнения, когда запрошенные данные уже переданы из вторичной в первичную память.

Кроме того, микросхема УУП содержит регистр состояния управления памятью (РСУП), чтобы определять дальнейшие шаги и состояние УУП во время текущей обработки информации. Введение этого регистра позволяет пользователю контролировать преобразование адресов, прерывания и следить за работой программы. Этот регистр содержит такую информацию, как флаг класса ошибки, флаг, отражающий ошибку преобразования, разряд номера прерывания, разряд, указывающий направление ошибочных данных, разряд направления прерывания, флаг ошибки, флаг прерывания, разряд использования преобразования пользователем, разряд использования преобразования супервизором и т. д.

Тенденции развития виртуальной памяти и кэш-памяти для микропроцессоров

Идеи организации виртуальной памяти с разбиением на сегменты и страницы, преобразованием адресов и защитой памяти применялись в устройствах управления памятью первого и второго поколений в 8- и 16-разрядных микропроцессорах. До сих пор для микропроцессоров виртуальная память строилась на одной микросхеме процессора и одной микросхеме устройства управления памятью. Однако постоянное совершенствование технологии приведет к тому, что 8-, 16- и 32-разрядные процессоры будут строиться со встроенной системой виртуальной памяти, со встроенным кэшем команд, кэшем для преобразования адресов и кэшем данных. Опять же для первого поколения таких микросхем и микроЭВМ будут использоваться концепции, хорошо известные по применению их в универсальных и мини-ЭВМ.

С одной стороны, будут предприниматься попытки расположить все эти элементы на одной микросхеме (для большинства применений). С другой стороны, для создания более мощных, быстродействующих и дешевых универсальных ЭВМ и мини-ЭВМ будут разрабатываться биполярные быстродействующие узлы, такие, как буфер меток и буфер данных. Для микроЭВМ такой подход будет использоваться при создании каскада микросхем для построения кэш-памятей более производительных и бóльшего объема. В ближайшем будущем кэш-память и виртуальная кэш-память станут обычными почти во всех микропроцессорах.

ЛИТЕРАТУРА

A1. Anderson J. P., Glaser E. L., Automatic Utilization of Hierarchical Memories, Proc. Gigacycle Computing Systems, AIEE Winter Meeting, Jan. 1962, pp. 124—132.

A2. Agrawal O. P., Cache Memory Systems for Multiprocessor Architecture, Proc. 1977 Nat. Compt. Conf., June 1977, pp. 13—16.

A3. Agrawal O. P., Zingg R. J., Pohm A. V., Proc. 14th IEEE Computer Soc. Int. Conf., IEEE, New York, 1977, p. 74.

A4. Amstadter B. L., Reliability Mathematics Fundamentals: Practices, Procedures, McGraw-Hill, New York, 1971.

A5. Allen C. A., Design of Digital Memories That Tolerate All Glasses of Errors, Ph. D. dissertation, Stanford University, Palo Alto, Calif., 1966.

A6. Agrawal O. P., Applicability of Cache Memories to Unconventional Computing Structures like SYMBOL-ZR, Ph. D. dissertation, Iowa State University, Ames, Iowa, 1974.

A7. Astrahan M. M., Rochester N., The Logical Organization of the New IBM Scientific Calculator, Proceedings of the Electronic Computer Symposium IRE, Los Angeles, Apr. 30-May 2, 1952, paper xix, p. 7.

A8. Anderson Judith A., Lipoveki G. J., A Virtual Memory for Microprocessors, IEEE 2nd Symposium on Computer Architecture, 1975, pp. 80—84.

B1. Bloom L., Cohen M., Porter S., Considerations in the Design of a Computer with High Logic to Memory Speed Ratio, Proc. Gigacycle Computing Systems, Jan. 29-Feb. 2, 1962, AIEE Special Publ. 2—136, pp. 53—63.

B2. Belady L. A., A Study of Replacement Algorithms for a Virtual Storage Computer, *IBM Systems Journal*, **5**, № 2, 1966, pp. 78—101.

B3. Bell C. G., Casasent D., Implementation of a Buffer Memory in Minicomputers, Computer Design, Nov. 1971, pp. 83—89.

B4. Bell J., Casasent D., Bell C. G., An Investigation of Alternative Cache Organizations, *IEEE Trans. Computers*, **C-23**, Mar. 1974, p. 346.

B5. Baum A. and others, Hardware Considerations in a Micro-Computer Multiprocessing System, Compcon, 1975, Spring Digest, Feb. 1975, pp. 27—30.

B6. Bhandarker D. P., Analysis of Memory Interference in Multiprocessors, *IEEE Trans. Computers*, **C-24**, № 9, Sept. 1975.

B7. Baer J. L., A Survey of Some Theoretical Aspects of Multiprocessing, *ACM Computing Surveys*, **5**, № 1, Mar. 1973, pp. 31—80.

B8. Baylis M. H. J., Fletcher D. G., Howarth D. J., Paging Studies Made on the I. C. T. Atlas Computer, Proc. Int. Federation Information Processing Congr., Edinburgh, 1968, North-Holland, Amsterdam, 1968, p. D113.

B9. Belady L. A. Kuehner C. J., Dynamic Space Sharing in Computer Systems, *Comm. ACM*, **12**, 1969, pp. 282—288.

B10. Barnes D., Bursky D., As Memory Density Quadruples Again, Designers Focus on Reliability, Electronic Design, Jan. 18, 1979, pp. 34.

B11. Brainerd J. G., Sharpless T. K., The ENIAC, Electrical Engineering, 1967 (Feb. 1941), p. 163.

B12. Buchholz W., Design Objectives for the IBM Stretch Computer, AFIPS Proc. Western Joint Computer Conf., 1957, pp. 99—105.
B13. Basche C. J., and others, The Architecture of IBM's Eraly Computers, *IBM J. Res. Dev.*, **25**, № 5, Sept. 1981, pp. 363—375.
B14. Bloch Erich, The Engineering Design of the Stretch Computer, Proc. the Eastern Joint Computer Conf., 1959, pp. 48—56.
B15. Brooker R. A., Some Techniques for Dealing with Two Level Storage, *Computer Journal*, **2**, 1960.
B16. Beckmann P., Probability in Communications Engineering, Harcount Brace Jovanovich, New York, 1967.
C1. Conti C. J., Concepts for Buffer Storage, *Computer Group News*, **2**, Mar. 1969, pp. 9—13.
C2. Carter W. C., Jessep D. C., Wadia A., Error-Free Decoding for Failure-Tolerant Memories, Proc. IEEE Computer Group Conf., June 1970.
C3. Carter W. C., Duke K. A., Jessep D. L., Jr., Lookaside Techniques for Minimum Circuit Memory Translaters, *IEEE-IC*, **C-22**, № 3, Mar. 1973.
C4. Cork M. L., Reliability with Error-Detecting and Correcting Codes in Semiconductor Memories, Ph. D. dissertation, University of Arizona, Tucson, 1975.
C5. Chu W. W., Opderbeck H., Performance of Replacement Algorithms with Different Pay Sizes, *Computer*, **7**, Nov. 1974, pp. 14—21.
C6. Censier L. M., Feautrier P., A New Solution to Coherence Problems in Multicache Systems, *IEEE Trans. Computers*, **C-27**, № 12, Dec. 1978, pp. 1112—1118.
C7. Conti C. J., Gibson D. H., Pitkowsky S. H., Structural Aspects of the System/380 Model 85: General Organization, *IBM Systems Journal*, **7**, № 1, 1968, pp. 2—13.
C8. Chow C. K., Determination of Cache's Capacity and Its Matching Storage Hierarchy, *IEEE Trans. Computers*, **C-25**, № 2, Feb. 1976, pp. 157—164.
C9. Chow W. M., Central Server Model for Multiprogrammed Computer Systems with Different Classes of Jobs, *IBM Res. Dev.*, **19**, № 3, May 1975, pp. 314—320.
C10. Callahan J., Patel C. N., Stevenson D., Bringing Virtual Memory to Microsystems, *Electronics*, June 30, 1981, pp. 119—122.
D1. Denning P. J., The Working Set Model for Program Behavior, *J. Assoc. Computer Mach.*, **1**, May 1968, pp. 323—333.
D2. Data General, New Eclipse Handles Demanding Real Time Applications, *Data General News*, **1**; № 1, Feb. 1975, pp. 1—4.
D3. Data General, Front-ending MOS Memories with Bipolar Cache: Best of Both Worlds, Input/Output, *Data General Users Group*, **2**, № 2, 1975, pp. 6—8.
D4. Digital Equipment Corp., Translation Buffer, Cache, and SBI Control Technical Description, VAX-11/780 Implementation, EK-MM780-TD-001, 1978.
D5. Data General, Eclipse MV/1000 Principles of Operation, doc. № 014-00E48, 1980.
D6. Digital Equipment Corp., Computer Engineering, 1979, pp. 263—266.
D7. Denning P. J., Equipment Configuration in Balanced Computer Systems, *IEEE Trans. Computers*, **C-18**, Nov. 1969, pp. 1008—1012.
E1. Enslow P.H., Jr., Multiprocessors and Parallel Processing, Wiley, New York, 1974.
E2. Easton M. C., Model for Interactive Data Base Reference String, *IBM J. Res. Dev.*, **19**, Nov. 1975, pp. 550—556.
E3. Eckert J. P. and others, Design of Univas-LARC System I, AFIPS Proc., Eastern Joint Computer Conference 1959, pp. 59—63.
E4. Eckert J. P., Univac-LARC, the Next Step in Computer Design, Proc. 1956 EJCC (AIEE special publication T-107), pp. 16—19.

F1. Fotheringhan J., Dynamic Storage Allocation in the ATLAS Computer Including an Automatic Use of a Backing Store, *Comm. ACM*, 4, 1961, pp. 435.
F2. Ferris-Prabhu A. V., Improving Memory Reliability through Error Correction, Computer Design, July 1979.
F3. Fuller S. H., Baskett F., An Analysis of Drum Storage Units, *J. Assoc. Computer Mach.*, **22**, Fall 1975, pp. 83—105.
G1. Gibson D. H., Considerations in Block-Oriented Systems Design, Proc. AFIPS Spring Joint Computer Conf. **30**, 1967, pp. 75—80. Spartan, Washington, D. C.
G2. Gibson D H., Shevel W. L., Cache Turns up a Treasure, *Electronics*, Oct. 13, 1969, pp. 105—107.
G3. Goldberg J., Levitt K. W., Wensley J, An Organization for a Highly Survivable Memory, *IEEE-IC*, **C-23**, № 7, July 1974, pp. 693—705.
G4. Goldstine H. H., Goldstine A., The Electronic Numerical Integration and Computer, Math. Tables Aids Computers, **2**, July 1946, 97.
G5. Glorioso R. M., Chase T. D., Design of Virtual Memory for Small Computers, *Computer Design*, **12**, 1973, pp. 67—72.
G6. Gelenbe E., The Distribution of a Program in Primary and Fast Buffer Storage. *Comm. ACM*, **16**, № 7, July 1973, pp. 431—434.
G7. Gecsei J., Lukes J. A., A Model for the Evaluation of Storage Hierarchies, *IBM Systems J.*, № 2, 1974, pp. 163—178.
G8. Gecsei J., Determining Hit Ratios for Multilevel Hierarchies, *IBM J. Res. Dev.*, July 1974, pp. 318—327.
H1. Hamming R. W., Error Detecting and Error Correction Codes, *Bell System Tech. J.*. Apr. 1950, pp. 147—160.
H2. Hill F. J., Peterson G. R., Introduction to Switching Theory and Logical Design, Wiley, New York, 1966.
H3. Hellerman H., Digital Computer System Principles, McGraw-Hill New York, 1973.
H4. Haltfield D. J., Experiments on Page Size, Program Access Patterns, and Virtual Memory Performance, *IBM J. Res. Dev.*, Jan. 1972, pp. 58—66.
I1. Inkol R. J., Chamberlain S. G., Design and Realization of a Two-Level 64K Byte CCD Memory System for Microcomputer Applications, *IEEE J. Solid State Circuits*, **SC-15**, № 1, Feb 1980, pp. 131—135.
J1. Joseph M., An Analysis of Storage Hierarchies in Digital Computers, Ph. D. dissertation, Churchill College, University of Cambridge. June 1968.
J2. Joseph E. C., Innovations in Heterogeneous and Homogeneous Distributed Functions Architecture, *Computer*, Mar. 1974, pp. 17—24.
J3. Juliussen J. E., Mowle F. J., Multiple Microprocessors with Common Main and Control Memories, *IEEE Trans. Computers*, **C-22**, № 11, Nov. 1973, pp. 999—1007.
J4. Johnson R. C., Microsystem Exploit Mainframe Method, *Electronics*, Aug. 11, 1981, pp. 119—137.
K1. Kroeger J. H., Meade R. M., Cache Buffer Memory Specification, Proc. 1971 Computer Designers Conf., **1**, 1971.
K2. Kaplan K. R., Winder R. O., Cache-Based Computer Systems. *Computer*, Mar. 1973, pp. 30—36.
K3. Kurtzberg J. M., On the Memory Conflict Problem in Multiprocessor Systems, *IEEE Trans. Computers* **C-23**, № 3, Mar. 1974, pp. 286—293.
K4. Koppel R., RAM Reliability in Large Memory Systems — Significance of Predicting MTBF, *Computer Design*, Feb. 1979, p. 148.
K5. Koppel R., RAM Reliability in Large Memory Systems — Improving MTBF with ECC, *Computer Design*, Mar. 1979, p. 196.
K6. Katzan H., Storage Hierarchy Systems, AFIPS Proc. Spring Joint Computer Conf., 1971, pp. 325—336.

K7. Kilburn T. and others, Digital Computers at Manchester University, *Proc. IEEE* (London), **100**, 1953, p. 487.
K8. Kilburn T. and others, One-Level Storage Systems, *IRE Trans. Computers*, **EC-11**, 2, Apr. 1962, pp. 223—235.
K9. Kuhn L., NSL6000 Brings Benefits of Virtual Memory to Microprocessors, WOS Conf. Anaheim, Sept. 16—18, 1950, pp. 1—7, Prentice-Hall, Englewood Cliffs, N. J.
L1. Lee F. F., Project MAC Summer Study, Project MAC Report, Memorandum MAC-M-99, August 1963.
L2. Lee F. F., Look-Aside Memory Simulation, Project MAC Report, Memorandum MAC-M-131, Jan. 1964.
L3. Liptay J. S., Structural Aspects of the System, 360/65:11, The Cache. *IBM Systems J.*, **7**, № 15, 1968.
L4. Lee F. F. Study of Look-Aside Memory, *IEEE Trans. Computers*, **C-13**, № 11, Nov. 1969, pp. 1062—1064.
L5. Lonsdale K., Werburton E., Mercury: A High-Speed Digital Computer, *Proc. IEE*, **103**, pt. B, suppl. 2, 1956, pp. 174—183.
L6. Lavi Y., and others, A Bit Microprocessor Enters Virtual Memory Domain, *Electronics*, Apr. 16, 1980.
L7. Lin Y. S., Mattson R. L., Cost Performance Evaluation of Memory Hierarchies, *IEEE Trans. Magnetics*, Sept. 1972, pp. 390—392.
L8. Lin S., Introduction to Error Correcting Codes, Prentice-Hall, Englewood Cliffs, N. J., 1970.
M1. Meade R. M., On Memory System Design, Proc. AFIPS Fall Joint Computer Conf., 1970, pp. 33—43.
M2. Meade R. M., Design Approaches for Cache Memory Control, *Computer Design*, Jan. 1971, pp. 87—92.
M3. Meade R. M., How a Cache Memory Enhances a Computer's Performance, *Electronics*, Jan. 17, 1972, pp. 58—63.
M4. Mitchell J. and others, Multiprocessor Performance Analysis, Proc. National Computer Conf. **43**, May 1974, pp. 399—403.
M5. Mackinnon R. A., Advanced Function Extended with Tightly Coupled Multiprocessing, *IBM Systems J.*, **13**, № 1, 1974, pp. 32—59.
M6. Monroe R. N., Add-in Cache Memory Doubles Minicomputer Processing Speed, *Computer Design*, Oct. 1979, pp. 115—120.
M7. Mudge J. C., Design Decisions Achieve Price/Performance Balance in Mid-Range Minicomputers, *Computer Design*, Aug. 1977, pp. 87—95.
M8. Mattson R. L. and others, Evaluating Techniques for Storage Hierarchies, *IBM Systems J.*, **9**, № 2, 1970, pp. 78—117.
M9. Montgomery R. C., Simple Hardware Approach to Error Detection and Correction, *Computer Design*, Nov. 1978, pp. 109—118.
M10. Mattson R. L., Evaluation of Multilevel Memories, *IEEE Trans. Magnetics*, **MAG-7**, Dec. 1971, pp. 814—819.
M11. Matick Richard, Computer Storage Systems and Technology, Wiley, New York, 1977.
N1. Nessett D. M., *Aust. Computer J.*, **7**, № 33, 1975.
N2. Noguchi K., Ohnisi L., Morita H., Design Considerations for a Heterogeneous Tightly Coupled Multiprocessor System, Proc. National Computer Conf., **44**, May 1975, pp. 551—559.
N3. National Semiconductor NS16082 MMU preliminary data sheet, 1982.
P1. Peterson W. W., Weldon E. J., Jr., Error Correcting Codes, MIT Press, Cambridge, Mass., 1972.
P2. Pohm A. V., Cost/Performance Perspectives of Paging with Electronic and Electromechanical Backing Stores, *Proc. IEEE*, **63**, № 8, Aug. 1975, pp. 1123—1128.
P3. Pohm A. V., Agrawal O. P., Monroe R. N., The Cost and Performance Pradeoffs of Buffered Memories, *Proc. IEEE*, **63**, Aug. 1975, pp. 1120—1135.

P4. Pohm A. V., Argawal O., Cheng C., Fabritek Buffered Memory Study, Engineering Research Institute, Iowa State University, Ames, Iowa, 1973.
P5. Pucknell D. A., A Preliminary Study of Cache Storage for the FM2000 CPU, Ferranti Digital Systems Division, Tech. Note C30/TN116.
P6. Pucknell D. A., Costs Effectiveness Method for Speeding up Digital Computer Hardware, IREE (Australia), International Electronics Convention, Sydney, Aug. 1975, Convention Digest, pp. 7—9.
P7. Papian W. N., High-Speed Computer Stores 2,5 Megabits, *Electronics*, **30**, Oct. 1957.
P8. Pohm A. V., Smay T. A., Computer Memory Systems, *Computer*, Oct. 1981, pp. 93—110.
P9. PDP 11/70 Processor Handbook, Digital Equipment Corporation, 1978.
P10. Peterson W. W., Error Correcting Codes, 2nd ed., MIT Press, Cambridge, Mass., 1961.
R1. Rao G. S., Performance Analysis of Cache Memories, *JAC*, **25**, № 3, July 1978, pp. 378—395.
R2. Raphel H. A., Distributed Intelligence Microcomputer Design, Comcon 75, Spring Digest, 10th Annual IEEE Computer Society International Conference, Feb. 1975, pp. 21—26.
R3. Rupp E. J., Memory Management Chip Extendes Reach of 8 Bit Precisions, *Electronic*, Aug. 25, 1981, pp. 134—136.
R4. Rhodes C., Caches Keep Main Memory from Slowing Down Fast CPU's, *Electronic Design*, Jan. 21, 1982, pp. 179—184.
R5. Rhodes C., Chun J., Herndon T., Computer Memory Functions Surface on VLSI Chip, *Electronic Design*, Feb. 18, 1982, pp. 159—163.
S1. Scarott G. G., The Efficient Use of Multilevel Storage, Proc. IFIPS Congress, Spartan Books, Washington, D. C., 1965.
S2. Searle B. C., Froberg D. E., Tutorial: Microprocessor Applications in Multiple Processor Systems, *IEEE Computer*, Oct. 1975, pp. 22—30.
S3. Sastry K. V., Kain R. Y., On the Performance of Certain Multiprocessor Computer Organizations, *IEEE Trans. Computers*, **C-24**, № 11, Nov. 1975, pp. 1066—1074.
S4. Sanyal S., Venkataraman K. N., Single Error Correcting Code Maximizes Memory System Efficiency, *Computer Design*, May 1978, pp. 175—184.
S5. Lin Shu, An Introduction to Error Correcting Codes, Prentice-Hall, Englewood Cliffs, N. J., 1970.
S6. Sisson S. S., Flynn M. J., Addressing Patterns and Memory Handling Algorithms, AFIPS Proc. Fall Joint Computer Conf., **33**, 1968, pp. 957—967.
S7. Strecker W. D., Cache Memories for PDP. 11 Family Computers, Proc. 3th Annual Symp. Computer Architecture, 1976, pp. 155—158.
S8. Stevenson D., An Introduction to Memory Management, *Electronics and Power*, Apr. 1980.
S9. Slutz D. R., Traiger I. L., Determination of Hit Ratios for Class of Staging Hierarchies, IBM Research Report RJ 1044.
S10. Scherr A. L., Analysis of Computer Memory Performance, American Statistical Association Conference Session on Probability and Statistics in the Design of Computer Systems, Los Angeles, Aug. 18, 1966.
T1. Takahashi S. H., Nishino K., Fuchi K., System Design of the ETL MK-6 Computer, Proc. IFIPS Congress, 1962, North-Holland, Amsterdam, 1963, pp. 690—693.
T2. Tucker J. H., Automatically-Loaded Scratchpad Memories in Digital Computer Memory Systems, Ph. D. dissertation, Churchill College, University of Cambridge, Feb. 1968.
T3. Tang C. K., Cache System Design in the Tightly Coupled Multiprocessor Systems, *AFIPS Proc.*, **49**, 1976, p. 749.

T4. Traiger I. L., Mattson R. L., The Evaluation and Selection of Technologies for Computer Storage Systems, Proc. Conference on Magnetism and Magnetic Materials, № 5, part 1, pp. 1—12.

T5. Tate G., Miller W., EDC Chip Boosts Memory Reliability, *Electronic Design*, Sept. 1, 1980, pp. 151—155.

T6. Toshi E. A., Watanabe T. An All-Semiconductor Memory with Fault Detection Correction and Logging, *Hewlett-Packard J.*, Aug. 1976, pp. 8—13.

T7. Tredennick H. L., Welch T A., High-Speed Buffering for Variable Length Operands, Symposium on Computer Architecture, IEEE Comp. Soc., College Park, Md. Mar. 1977, pp. 205—210.

T8. Traiger I. L., Slutz D. R., One Pass Techniques for the Evaluation of Memory Hierarchies, IBM Research Report, RJ-892, 1971.

V1. Von Neumann J., Burks A. W., Goldstein H., Preliminary Discussion of the Logical Design of an Electronic Computing Instruments, in J. von Neuman, Collected Works, V., Pergamon Press, Elmsford, N. Y., 1963.

V2. Aken Jerry, Match Cache Architecture to the Computer System, *Electronic Design*, Mar. 4, 1982, pp. 73—91.

W1. Wilkes M. V., Slave Memories and Dynamic Storage Allocation, *IEEE Trans. Electronic Computers*, Apr. 1965, pp. 270—271, Project MAC-M-164, MIT, Cambridge, Mass., June 22, 1964.

W2. Whalen M. W., Speeding up Ferrite-Core Memories, *Electronics*, Oct. 13, 1969, pp. 108—110.

W3. Weisbecker J. A., Memory Systems, U. S. Patent № 3601812, Aug. 24, 1971.

W4. Winder R. O., A Data Base for Computer Performance Evaluation, представлена на IEEE Workshop on System Performance Evaluation, Argonne, Ill., Oct. 1971, и опубликована в *Computer*, **6**, № 3, Mar. 1973.

W5. Wiesen J. M., Mathematics of Reliability, Proc. 6th National Symp. Reliability Quality Control, Jan. 1960.

W6. Williams F. C., Kiliburn T., A Storage System for Use with Binary-Digital Computing Machines. *Proc. IEEE* (London), **46**, № 3, Mar. 1949, p. 81.

W7. West J. T., Lasting Computer Design Exploit Standard Parts, *Electronics*, Nov. 13, 1975, pp 130—136.

W8. Winograd S., On Parallel Evaluation of Certain Arithmetic Expressions, *J. ACM*, **22**, № 4, Oct. 1975, pp. 477—492.

Y1. York K. D., A Low-Cost High-Performance Alternative to Fully Associative Buffer Memories, Private Memo, Architecture Department, Burroughs Corp., Computer Systems Group, July 11, 1973.

Z1. Zilog, An Introduction to the Z8010 MMU Memory Management Unit, Mar. 1981.

Z2. Zilog, Z8000 vs. 68000 Segmented vs. Linear Addressing, concept paper.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

Адрес базовый 242
— виртуальный 218, 219, 224—238
— логический 166, 183, 189, 199, 241, 243
— физический 29, 166, 199, 215, 218, 224—238
Адресация основной памяти 24
Адресное пространство 32, 224, 239
Алгоритм обратной записи в конфликтных ситуациях 38—40
— — — — — — с флагами 40, 41, 54—60, 104
— простого свопинга 35, 38—41, 44, 54—60
— регистрового свопинга с флагами 41, 54—60, 87, 104
— свопинга 30, 31, 35
— сквозного накопления 36—38, 54—60
— сквозной записи 36—39, 41, 44, 51—60, 92, 98

Бит контроля четности 136—142
Буфер с прямым отображением 33, 34
Быстродействие кэша 18, 31, 90
— памяти 7, 18, 23—28, 51, 61
— процессора 18, 30, 31
— эффективное буфера 30

Вероятность неудачного обращения 33—36, 42, 70—79, 83, 87, 99, 105
— — — за страницей 123—125
— удачного обращения 25—27, 31—33, 42, 50, 73—85
Время доступа процессора к памяти 11
— — среднее 11
— — эффективное 16
— обращения эффективное 29, 30
— поиска указателя 26
— цикла 7, 18, 25, 60
— — кэш-памяти 18, 25, 26
— — медленной памяти 26
— — процессора 11, 18, 30

Дескриптор 242—245
Длина кода 135, 137
Емкость буфера 30, 72, 78, 129
— кэша 18—20
— памяти 7—10, 18, 24, 129, 166, 169, 176, 178

Задержка для отдельного модуля (DS) 52
— средняя на один модуль (DA) 52—61
Запоминающее устройство 9, 25
— — ассоциативное 25, 32
— — буферное 23
— — быстродействующее 27
— — оперативное 24
— — полупроводниковое 133—150

Идентификатор 23
Иерархия аппаратно-управляемая 11
— памяти 9—21
— управляемая пользователем 11
— — системным обеспечением 11
Интенсивность отказов 143, 144

Код корректирующий 133, 149—151
— *Хемминга* 133, 136, 155—158
— с обнаружением одиночных ошибок 136
Кодовое слово 135
Коррекция ошибок 150—154
Коэффициент избыточности кода 135—137
Кэш адресный 18, 23
— данных 23, 219, 229
— именной 18
— память 9, 16, 22, 85, 88, 92, 101, 179
— программный 18, 23

Метод выборки 30
— замещения информации 30
— отображения 29
Микропроцессор AMD 28000 239
— IAPX 240
— Intel 8086 239
— Motorola 68000 239
— Z 8001 241—248
— Z 8003 248
Микросхема AMZ 8010 240
— MC 6829 248—250
— NS 16082 249—252
— Z 8001 240—245
— Z 8010 240—248
Мини-ЭВМ ЭКЛИПС 214, 215
— MV/8000 222, 232—238
— PDP-1 178
— — 11/34 246, 247
— — 11/45 15
— — 11/60 17, 221, 222
— — 11/70, 15, 17, 215—222
— — 8E 212
— VAX-11/780 222—232

Операционная система 233, 240
Отображение адресов 30

Память ассоциативная 16, 23
— буферная 22, 24, 177
— — с прямым отображением 33
— виртуальная 15—17, 79, 123
— вспомогательная 14, 173, 174
— медленная 26
— многоклассовая с буфером 33, 34
— многомодульная 51, 52, 87
— основная 17, 35—44, 94
— поисковая 32
— полупроводниковая 13, 133
— признаковая 23
Правило выборки по запросам 30
— — с упреждением 30
Преобразование адресов 225—227, 182—185
Приоритетный список обновления данных 23, 25
Присвоение адресов 35
Пропускная способность шин 89, 91, 100—104

Распределение памяти 14, 15
— — динамическое 14, 15, 173
— — статическое 14, 15
Расстояние *Хемминга* 135, 137
Регистр адресный 166, 167, 171, 180
— базового адреса 246, 247
— базовый 174
— буфера данных 171
— данных входной 155, 156
— — выходной 156—158
— диагностический 156, 158
— достоверности 181
— индексный 16 ,165, 170, 173
— контрольных битов входной 156, 157
— признаков 174, 175
— процессора 15
— сдвига 25
— управляющий 173

Связность информации 89
— системы памяти 89, 90
Система виртуальной памяти с буфером 17
— — — — кэшем 17, 18
— — — со страничной организацией 17
— мажоритарная 143, 147—149
— мультикэш-процессорная 91
— мультипроцессорная 88—108, 120—123
— памяти буферная 25—28

— — иерархическая 11, 12
— — многоуровневая 12
— параллельная 143, 147—149
— последовательная 143, 147—149
— с буферной памятью 15, 29, 42
— — виртуальной памятью 7
— — кэш-памятью 38, 179, 185
— — разделением времени 15, 51
Справочник 23, 32—42, 52, 90
Среднее время между отказами (СВМО) 146, 149
— — наработки на отказ (СВНО) 146—148, 151
Стек арифметический 16, 170
— памяти 171
— программный 16, 170
Стратегия замещения блоков «первым пришел — первым ушел» (FIFO) 68, 79
— — наименее используемого блока (LRU) 68, 79, 81, 108, 124, 130, 180
— — страниц 129, 131
Схема коррекции ошибок 138, 140
— преобразования адресов 29

Трафик шинный 91, 99

Устройство управления памятью 23, 215—217, 240—255

Центральный процессор 10, 35, 41, 179, 240

Частота использования блока 67—75
— — слов 69—75

ЭВМ ATLAS 15—16, 162—168, 179
— B6500 15
— CDC 7600 15
— ETL-Мк-6 16, 169, 170
— GE 645 15
— IBM 3033 197—203
— — 360/195 15, 187, 196, 197
— — 360/85 15, 17, 179—187
— — 360/95 11
— — 370/155 187—192
— — 370/158 15, 197
— — 370/165 187, 192—197
— — 370/168 15, 197
— STRETCH 162
— SYMBOL-2R 15, 209, 210
— UNIVAC I, IV 7, 8
Экспоненциальный закон изменения надежности 144
— — — вероятности отказа 144
Эффективная задержка 30
— стоимость памяти 7—28
Эффективное быстродействие буфера 30
— — кэша 90
— время выборки информации 60, 174
— — доступа 16, 29
— — обращения 29
— — цикла 25, 26, 29, 40—46, 60—65, 87, 105
Эффективность кода 135, 137

Язык высокого уровня 14, 240

ОГЛАВЛЕНИЕ

Предисловие редактора перевода 5
Предисловие . 6

Глава 1. Быстродействующие запоминающие устройства ЭВМ . . . 7

1.1. Введение . 7
1.2. Обзор запоминающих устройств и способов хранения информации 9
1.3. Прошлое и настоящее иерархий быстродействующей памяти . . 16
Упражнения . 21

Глава 2. Основы построения буферной памяти (кэш-памяти) 22

2.1. Введение . 22
2.2. Основные элементы кэш-памяти 23
2.3. Экономическая и эксплуатационная эффективность 23
Упражнения . 27

Глава 3. Проектирование систем с кэш-памятью 29

3.1. Введение . 29
3.2. Разделение буфера 32
3.3. Алгоритмы свопинга 35
3.4. Сравнение производительности систем памяти 42
3.5. Системы с многомодульной основной памятью 51
3.6. Эффективные времена цикла и выборки информации 60
Упражнения . 63

Глава 4. Вероятность удачного обращения и влияющие на нее факторы 67

4.1. Аналитические модели 67
4.2. Зависимость вероятности удачного обращения от размеров блока 75
4.3. Заполнение буфера и вероятность неудачного обращения в начале выполнения задания 79
Упражнения . 83

Глава 5. Иерархическая организация памяти 85

5.1. Введение . 85
5.2. Буферизация множества модулей памяти 85
5.3. Связность информации в кэш-памяти 89
5.4. Организация мультикэш-процессорной системы 91
5.5. Пропускная способность шин при обслуживании мультипроцессоров 101
5.6. Оптимизация трехуровневых иерархических структур 104
5.7. Пример проектирования кэш-памяти мультипроцессорной системы 108
5.8. Системы виртуальной памяти 118
Упражнения . 131

Глава 6. Корректирующие коды и надежность быстродействующих систем памяти . 133

6.1. Введение . 133
6.2. Коды, обеспечивающие обнаружение и исправление ошибок . . . 133
6.3. Анализ надежности 142
6.4. Влияние корректирующих кодов на надежность систем памяти . . 149
6.5. Серийно выпускаемые схемы коррекции одиночных ошибок . . 154
Упражнения . 159

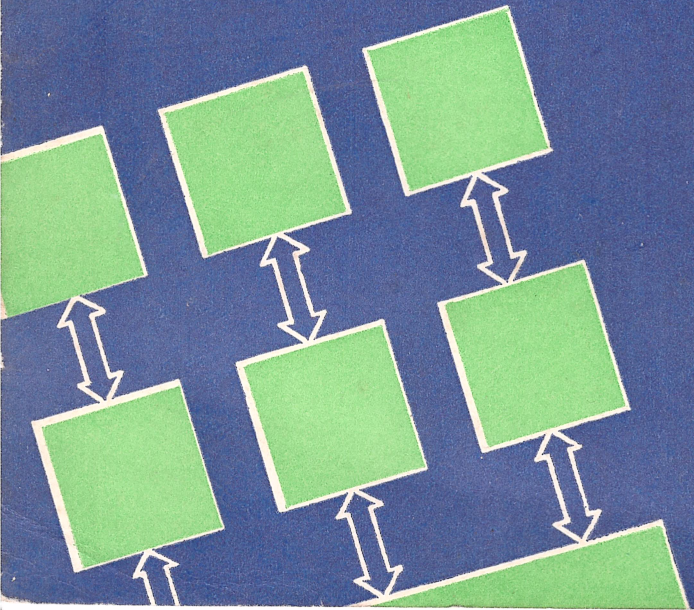
Глава 7. Эволюция иерархических систем памяти и систем с кэш-памятью . 161

7.1. История развития иерархических систем памяти 161
7.2. Большие системы с кэш-памятью. Системы с кэш-памятью фирмы IBM . 179
7.3. Другие вопросы, относящиеся к кэш-памяти 202
7.4. Кэш-память в мини-ЭВМ 212
7.5. Управление запоминающими устройствами и работа с кэш-памятью в микропроцессорах 239
Литература . 256
Предметный указатель 262

1р. 20 к.

БЫСТРО-ДЕЙСТВУЮЩИЕ СИСТЕМЫ ПАМЯТИ

Книга посвящена вопросам расчета и проектирования быстродействующих систем памяти. Рассматриваются различные структуры памяти, процедуры обмена информацией между уровнями, способы преобразования адресов. Подробно описаны быстродействующие системы памяти машин общего назначения, мини-и микро ЭВМ, микропроцессорных устройств.



БЫСТРОДЕЙСТВУЮЩИЕ СИСТЕМЫ ПАМЯТИ

ИЗДАТЕЛЬСТВО МИР